DICK SMITH ELECTRONICS

VZ300

PERSONAL COLOUR COMPUTER

MAIN UNIT MANUAL

LIMITED WARRANTY

Video Technology Ltd. shall have no liability or responsibility to purchaser or any other person or entity with respect to any liability, loss or damage caused or alleged to be caused directly or indirectly by this product, including but not limited to any interruption of service, loss of business or anticipatory profits or consequential damages resulting from the use or operation of this product.

FIRST EDITION — 1985

VERSION 2.0 Revision Zero — April 1984

INTRODUCTION

This manual is intended for people who want to learn to program in BASIC on the Colour Computer. With a little time and effort you will very soon discover that there is nothing very difficult about learning how to program your computer. You will be introduced to the fundamentals of BASIC and to the procedures of programming. Nothing is taken for granted. No prior knowledge is presumed. Things are explained step by step. All you have to do is to start at the beginning and make sure you try everything as it comes up. Take your time. Understand one step before going to the next.

The key to success is to try everything. It is not enough to read about it. You must do it. You don't learn to play the piano, type or swim by reading a book. You learn by doing. Don't worry about making mistakes. It is part of the learning process. If you do make a mistake, just correct the mistake and continue. The computer doesn't worry about it, why should you? There is nothing that can be done from the keyboard that can damage your computer. Cautions are included in the text when statements that might DESTROY data files are introduced. Thus, feel free to try things out with your computer at every stage of learning.

In general, you should follow the SEQUENCE of presentation given in the text. However, Chapter 15 which discusses the use of tape storage may be read at anytime when you wish to save a program on the cassette tape. While this manual is written for one who wishes to learn to program in BASIC on this Computer, it can also serve as a general introduction to programming in BASIC on any system. Just remember that the BASIC language has many forms. There are slight differences between one implementation of BASIC and another.

Finally, this manual will not only help you to understand BASIC but it will also help you to understand the fundamentals of computer programming in general.

Have fun with your colour computer!

PART I USER'S MANUAL

PART II BASIC REFERENCE MANUAL

PART III BASIC APPLICATION PROGRAMS

BOX CONTENTS

1. This Manual
2. Your Colour Computer.
3. A power adaptor.
4. A cord to connect your computer to the TV Antenna Outlet.
5. A cord to connect your computer to standard audio cassette recorder.
6. A demonstration tape.

To get your Computer working you will need a colour TV, a B/W TV or a DSE Monitor.

PART I

USER'S MANUAL

PROCEDURE FOR SETTING UP THE COLOUR COMPUTER

A. Disconnect the aerial cord from your television set. Connect a cord from the TV socket on your computer to the aerial socket on your TV set.

B. Connect your power adaptor to a normal wall AC outlet.

C. Connect the adaptor power plug to the power socket on your computer.

D. Switch on the power by pressing the power switch. Check the power indication lamp on the main console. If the power is on, there is a red light. If there is no red light switch off and check all your connections.

E. Turn on the television set and select the channel. The channel that you select should be one that you do not use for regular TV programs.

F. Tune to CHANNEL 1 on your VHF television set.

G. If the screen shows a green square with the word READY, then the set-up is all right.

H. If the screen does not seem to display a square, turn off the power of your computer and check all the connections.

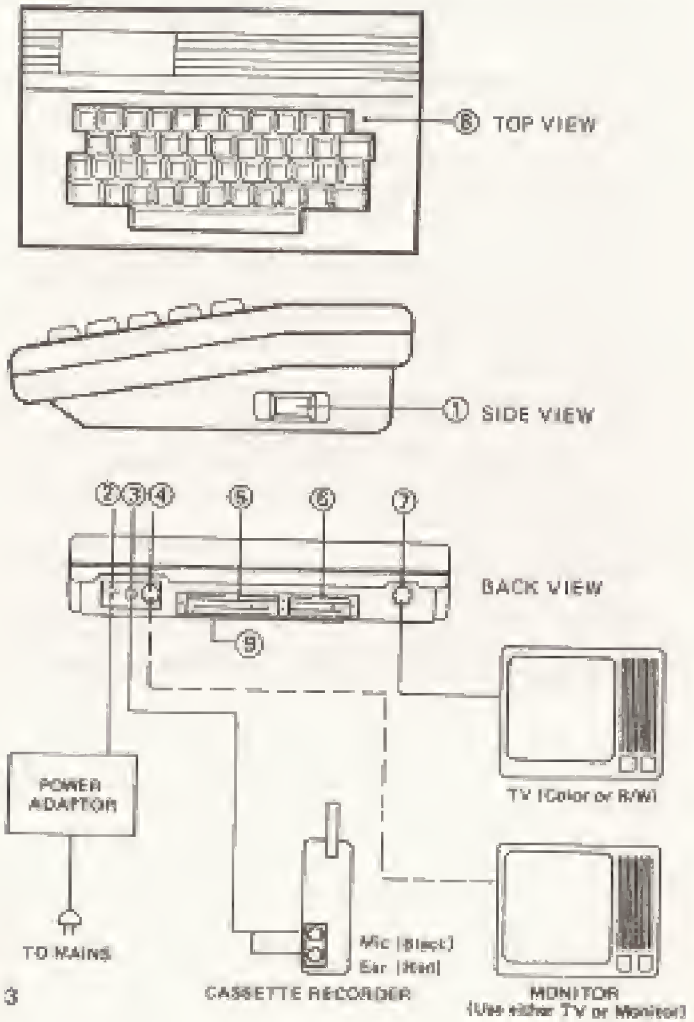
I. If the screen does not show the above message, switch off the power of your computer for about 10 seconds and switch on again.

J. If after several attempts you still do not succeed, switch off the power and check all the connections.

K. If a Monitor is connected, the mentioned message should come out when you turn on the power of your computer. If you cannot get this message, turn off the power of your computer and check all the connections. (also check the BRIGHTNESS level of the MONITOR)

L. Be sure to check the TV System of your computer. (TV or Monitor of PAL System should be connected to the computer PAL System.

DIAGRAM 1



CONNECTIONS AND SWITCH

1. **ON/OFF SWITCH**
   It controls the power supply to the computer, when it is switched on, a red indicator light (8) should show on the top right hand corner.

2. **POWER SOCKET**
   The cord from the power adaptor is connected to here.

3. **CASSETTE SOCKET**
   This is the cassette input/output socket. Your cassette tape recorder is connected to here.

4. **MONITOR SOCKET**
   If you are using a Monitor, then it is connected at this location.

5. **MEMORY EXPANSION**
   If you decide to use a memory expansion module with your computer, just remove the cover from this aperture and insert the memory expansion module. (BE SURE TO TURN OFF THE POWER BEFORE INSERTING OR REMOVING ANY EXPANSION MODULE)

6. **PERIPHERALS**
   Peripherals such as Printer Interface module or Joystick are connected at this location. (BE SURE TO TURN OFF THE POWER BEFORE INSERTING OR REMOVING ANY EXPANSION MODULE)

7. **TV SOCKET**
   The cord connecting the television antenna socket to your computer is connected to here.

8. **INDICATOR LIGHT**
   It indicates the power supply to your computer. When the power switch is turned on, it should glow.

9. **COLOUR DEFEAT SWITCH**
   For those using a green monitor, switch the selector to B/W position.

PRECAUTIONS

1. Keep the main unit away from liquids.
2. Avoid exposing the main unit to excessive heat. Store in a dry place.
3. Switch off the power and disconnect the power plug when not in use.
4. Do not drop the main unit. Handle it with care.
5. Make sure the power is turned off when inserting or removing any expansion modules such as Memory Expansion Module or Printer Interface Module.

TROUBLE SHOOTING CHECKLIST

SYMPTOM

1. No Indicator Light
   - The Mains supply not turned on or badly connected.
   - The Power Adaptor not properly connected or badly contacted.
   - Power switch not turned on.
   - The power plug badly connected
2. No Screen Display
   - Axial cord not properly connected or badly contacted.
   - The TV set not properly tuned.
   - Misconnection between the TV set and the Monitor socket.
   - Misconnection between the Monitor and TV socket.
3. Screen Display without the READY Signal
   - Improper set-up procedure, switch off the power for a while and switch on again.
4. Abnormal Performance
   - Improper set-up procedure. Switch off the power for a while and switch on again.
5. Cassette Loading and Saving Not Working
   - Cassette Interface Cord not properly connected or badly contacted.
   - The tone and volume level of the cassette recorder not set at proper range. (Please refer to Basic Programming Manual)

If none of the above work, please contact your nearest DSE Dealer.

The following message is valid for NTSC FCC version of Laser 310 only.

This equipment generates and uses radio frequency energy and if not installed and used properly, that is, in strict, accordance with the manufacturer's instructions, may cause interference to radio and television reception. It has been type tested and found to comply with the limits for a Class B computing device in accordance with the specifications in Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference in a residential installation. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

Reorient the receiving antenna

Relocate the computer with respect to the receiver

Move the computer away from the receiver

Plug the computer into a different outlet so that computer and receiver are on different branch circuits.

If necessary, the user should consult the dealer or an experienced radio/television technician for additional suggestions. The user may find the following booklet prepared by the Federal Communications Commission helpful:

'How to Identify and Resolve Radio-TV Interference Problems'.

This booklet is available from the U.S. Government Printing Office, Washington, DC 20402, Stock No. 004-000-00345-4.

This page is intended to leave bank

PART II

BASIC REFERENCE MANUAL

TABLE OF CONTENTS

CHAPTER 1

CHAPTER 2

CHAPTER 3

CHAPTER 4

CHAPTER 5

CHAPTER 6

CHAPTER 7

CHAPTER 8

CHAPTER 9

CHAPTER 10

CHAPTER 11

CHAPTER 12

CHAPTER 13

CHAPTER 14

CHAPTER 15

CHAPTER 16

CHAPTER 17

CHAPTER 18

CHAPTER 19

CHAPTER 1

THE COMPUTER

- WHAT IS A COMPUTER?
- WHAT MAKES UP A COMPUTER SYSTEM?
- WHAT IS A PROGRAM?
- COMPUTER LANGUAGES
- BASIC

WHAT IS A COMPUTER?

A computer is a device which performs various operations based on instructions given by the person who uses it. The computer cannot tell the user how to solve a problem. It has to be told what to do. The computer cannot think. Yet it is a very effective tool in the hands of a competent and experienced user.

A computer system consists of a number of machines or devices where operations are coordinated by a central control unit. These machines when working together are able to perform simple logical and arithmetic processes such as comparing two numbers. They can also read in information, store this information and give out results in a form understandable by us.

WHAT MAKES UP A COMPUTER SYSTEM?

Generally a computer system consists of the following units:

i) Central Processor Unit (CPU)
This can be considered the brain of the computer system. It performs operations specified in the instructions, such as arithmetic and logical operations.

ii) Memory Unit — Information and instructions given by the user or generated by the computer are stored here. This unit is inside the computer. The CPU gets information from it directly.

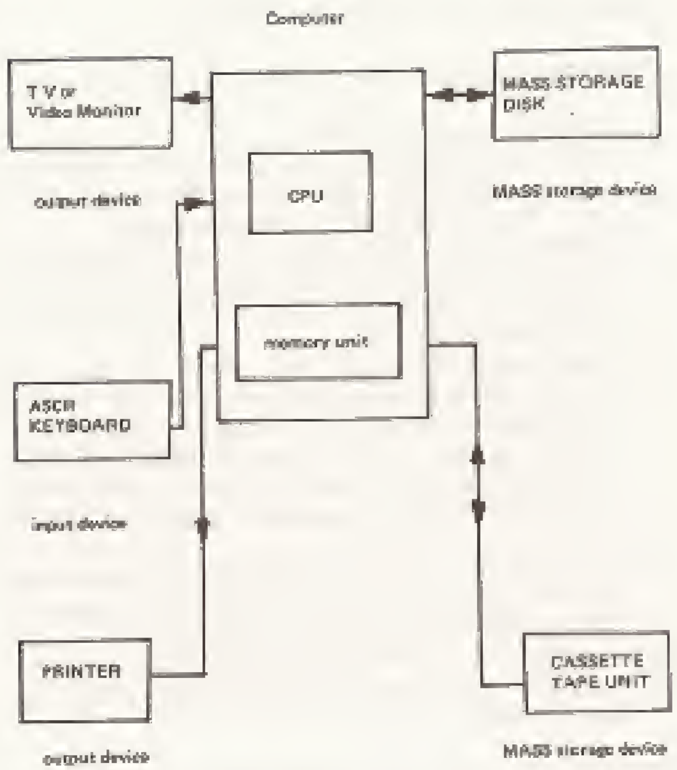
iii) Mass Storage Unit — This unit is outside the computer. It stores instructions and information given by the user or generated by the computer. The tape storage unit and the floppy disk units are examples of mass storage units. Information stored in these units has to be transferred to the internal memory unit before the CPU can process it.

iv) Input Device — As the name suggests this allows the user to enter instructions or information to the computer. The keyboard is an example of an input device.

v) Output Device — This receives information or results sent from the computer. Examples are the printer and the TV screen.

The input and output devices together act as a two-way communication channel between the computer user and the computer system.

Although computer systems vary in size, all practical computer systems require the above mentioned units.



Configuration of a Computer System in general

WHAT IS A PROGRAM?

A program is a set of instructions. The process of specifying a set of instructions for a computer is called programming. The individual preparing a program is called a programmer. The programmer feeds in or 'inputs' a series of instructions (the program) which tells the computer the steps to take to complete the task required of it.

COMPUTER LANGUAGES

There are two steps involved in preparing a program for a computer. First the programmer must know what instructions to specify and the order in which to specify them. Second, he must be able to communicate his instructions to the computer. Communication is accomplished by means of a programming 'language' which the programmer writes, and the computer 'reads'.

There are many programming languages in use today. Some are designed for very specialised applications. Others are designed for more general use. BASIC is a language in the latter category.

BASIC

BASIC, an acronym of Beginners' All-Purpose Symbolic Instruction Code, is a powerful programming language. BASIC has a simple English vocabulary, few grammatical rules and it resembles ordinary mathematical notation. To instruct your computer you must know BASIC. It will be introduced gradually and explained at each step.

Programs written in BASIC are translated by a language translation program into a language that the central processor unit understands. This language translation program is called the BASIC Interpreter, and is contained in the main console.

CHAPTER 2

HOW TO USE YOUR COLOUR COMPUTER

- TO START
- HOW TO OPERATE THE KEYBOARD
- THE PRINT COMMAND
- SYNTAX ERRORS
- EDITING
- INSERT
- CLS
- A LOOK AHEAD

TO START

When you have set up your colour computer and switched it on, your TV screen should look like this.

READY tells you that the computer is waiting and as the word suggests is ready to receive your instructions. The flashing square, the CURSOR, tells you exactly where you are on the screen. That is, where any information you feed in from the keyboard will appear on the screen.

The computer has 2 character display modes. One is the black character mode, as you can see on the screen now. Another is the green character mode. The character display modes can be selected by 2 methods.

The first method is very simple. Switch off your colour computer and then keep holding the CTRL key when you switch on the computer again. Now you can see the green character display mode on the screen.

The second method is to key in commands to the computer. At this moment, you may just type in these commands word by word. Later you will see how these commands work.

The black character display mode can be selected by typing:

POKE 3Ø744, Ø RETURN

The green character display mode can be selected by typing:

POKE 3Ø744, 1 RETURN

Now, try these commands to see the effect.

Note: RETURN means pressing the key labelled "RETURN" once.

HOW TO OPERATE THE KEYBOARD

The keyboard of your computer looks complicated but it is really not too difficult to operate. Most of the keys should look like:

Most keys comprise of 4 labels, LABEL 1 to LABEL 4. Each label can be typed by different methods.

LABEL 1 – Just press the key and get LABEL 1.

LABEL 2 – Press the key and the SHIFT key simultaneously.
Graphics characters are typed by this method.

LABEL 3 – Press the key and the CTRL key simultaneously.
This is the powerful statement entry. Some labels with the inverse characters are those commands that execute immediately but without a message appearing on the screen.

LABEL 4 – First press the CTRL and RETURN keys simultaneously. This will enter the FUNCTION mode. Then press the appropriate key and the CTRL key simultaneously.

Let us take the 9 key as an example. To get the number 9 just press the key. If you wish to get READ, hold down the CTRL key and at the same time press the 9 key. If you wish to get the bracket on the 9 key, hold down the SHIFT key and at the same time press the 9 key.

To get ABS (, which is below the 9 key, is a little more complicated. First press the CTRL key and hold it down while you press the RETURN key once and then press the 9 key. ABS (will appear on your screen. All it takes is a little practice. If you are wondering "What would happen if I did. . . . ." go ahead and do it and find out for yourself.

This page is intended to leave bank

THE PRINT COMMAND

You instruct your computer using the language of BASIC. The computer can obey at once, or it can store the instructions and run them later as a program. Let us now instruct the computer to act at once. To do this, we need our first word from the BASIC language: **PRINT**

Note: In looking at the operation of **PRINT** as our first BASIC command, we will be using the $\boxed{\text{RETURN}}$ key near the top right-hand corner of the keyboard. This key is used in BASIC to let the computer know when we have finished feeding in a command, so that it can go ahead and either carry out the command, or store it away in its memory to run later. Do not confuse this $\boxed{\text{RETURN}}$ key with the BASIC command **RETURN** (above the $\boxed{\text{R}}$ key on the keyboard), which we will meet later.

When using PRINT or any other statements you can type out each letter, e.g. P, R, I, N, T or you can just press the appropriate combination of keys, in this case [CTRL] and [P] (CTRL-P) to get the Keyword. Try it out on your computer.

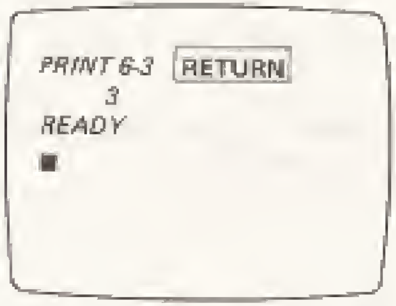
Note: (CTRL-P) means that you press the [CTRL] and [P] keys simultaneously.

Whichever way you type in **PRINT** the computer knows that it has to print what follows on the screen. For simplicity, you can just type'?' to represent the command **PRINT**.

For example, type *PRINT 6-3* and the screen should look like this.

Notice that when you press a key there is a (BEEP) sound. This tells you that the key has registered and is helpful in that you do not constantly have to check the screen.
Moreover, if you keep holding a key (or keys), the same character (or characters) is sent to the computer until you leave that key (or keys). If the total number of characters you typed in exceeds 64, the key-repeating function will stop. This is because the maximum number of characters that each statement can contain is 64.

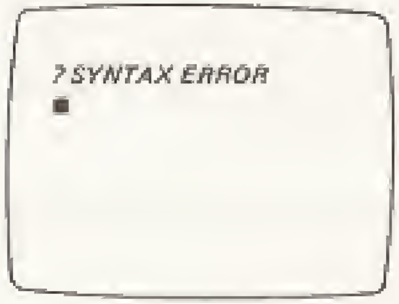
Now press [RETURN] and your screen should look like this.



By pressing the [RETURN] you have told the computer that the message is completed and you want the line executed. Remember then to press [RETURN] after each completed message.

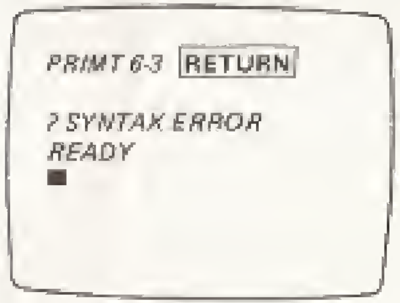
SYNTAX ERRORS

You may find the following appearing on your screen.



This means **SYNTAX ERROR.** A syntax error is usually due to incorrect punctuation or a typing error.

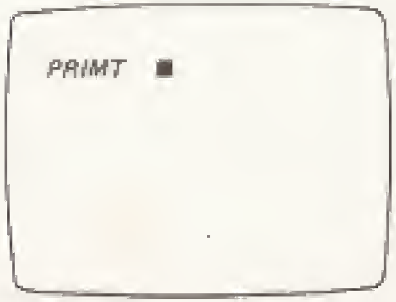
Suppose you type in *PRIMT 6.3* and then press the RETURN key your screen will look like this.



In addition to SYNTAX error there are a number of other errors that may occur. The various error types are listed in an appendix. These error messages tell you the reasons why your programs go wrong. If you are familiar with these messages, you can make use of them to correct your programs quickly and successfully.

EDITING

If you make a mistake while you are entering a program statement, you can use the CTRL and the appropriate key to move the cursor back to the wrong entry to make a correction.



First move the cursor over T by pressing ← (CTRL-M). Press RUBOUT (CTRL-;) and T will disappear. Move the cursor over M by ← (CTRL-M). Depress RUBOUT (CTRL -;) again and M will disappear. Then type in NT to get *PRINT*.

More generally, you can make corrections to your program lines using the cursor movement arrow keys. Also use the INSERT (CTRL-L) and the RUBOUT (CTRL-;).

There are four directions in which you can move the CURSOR: left, right, up and down as indicated by the black arrows on the keyboard. You will find these at the SPACE bar and the bottom right hand corner of your keyboard.

Let us take an example. If you have typed in a line, then the CURSOR is at the right side of the screen. Let us suppose you have made a mistake at the beginning of the line. You want to delete this line. Press ← (CTRL-M). Keep pressed until the CURSOR has moved back to where you want it. In this case, the character 'H'. Then press the RUBOUT (CTRL -;) and keep pressed until the line is erased.

If you want to eliminate *JOHN,* move the CURSOR to J by [←] (CTRL-M). Press [RUBOUT] (CTRL-;). Keep pressed and see what happens.

It is also possible to type in a letter over another letter. Let us suppose you want to change *JOHN* to *JAMES.* Position the CURSOR over O by pressing [←] (CTRL-M). Type in AMES and you get *JAMES.*

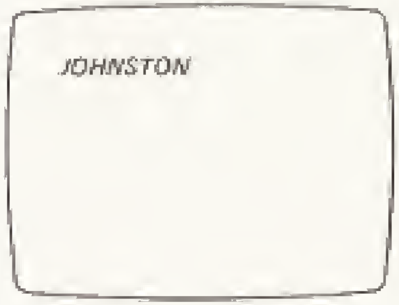
Suppose that after having typed *HELLO JAMES* you decide to change the name again. However this time suppose the CURSOR is on a lower line. Well all you do is to press [↑] (CTRL- .). You will find that the CURSOR will move up to the line you want to change. Each time you press [↑] (CTRL- .), the CURSOR will move up one line. Once you have reached the line you wish to edit you can just carry on the same way as the example above.

To familiarise yourself with editing you need to experiment with it. Here as elsewhere the old adage holds true: Practice Makes Perfect.

INSERT

This allows you to insert characters starting at the position the CURSOR is in without changing what is already there. For example: you wish to insert S in *JOHNTON* between the N and the T.

Well you move the CURSOR to the T press [INSERT] (CTRL-L) (which will put you into the INSERT Mode) and then type S. Your display should now look like this:



Be sure to press [RETURN] after you finish editing. This will update the current line where the CURSOR is located. This is particularly important with numbered program lines. If you forget to do so, the original line is still kept in the program.

CLS

As you have guessed by now [CTRL] means control. If you want to clear the whole screen, press [CLS] (CTRL-H) and then press [RETURN] . This will clean the screen but it will not clean the memory. The program will be wiped out from memory if you press [NEW] (CTRL-8). It will also be wiped out if you disconnect the power from the computer.

A LOOK AHEAD

At this stage you are probably very eager to jump ahead and see what your computer is capable of. So using your newly acquired ability try typing in these programs.

Be careful to type in everything. Do not worry about understanding the commands at the moment.

1) To get all the characters on the screen type this.

Example:

```
10  FOR I = 0 TO 1  [RETURN]
20  FOR J = 0 TO 255  [RETURN]
30  POKE 28672 + I*256 + J, J  [RETURN]
40  NEXT  [RETURN]
50  NEXT  [RETURN]
60  GOTO 60  [RETURN]
    RUN  [RETURN]
```

To stop this program, press [BREAK] (CTRL—).

2) To see some of the colour possibilities: try this one.

Example:

```
10  MODE (1)  RETURN
20  FOR Y = Ø TO 15  RETURN
30  FOR X = 1 TO 4  RETURN
40  COLOR X  RETURN
50  FOR K = Ø TO 31  RETURN
60  SET (X*32 – 32 + K, Y)  RETURN
70  NEXT  RETURN
80  NEXT  RETURN
90  NEXT  RETURN
100 GOTO 10  RETURN
    RUN  RETURN
```

To return to Text mode, press BREAK (CTRL—).

CHAPTER 3

YOUR COLOUR COMPUTER AS A SIMPLE CALCULATOR

- SIMPLE INSTRUCTIONS
- ORDER OF NUMERIC OPERATIONS
- BRACKETS

SIMPLE INSTRUCTIONS

To use the computer as a calculator simply type **PRINT** followed by the problem and then press [RETURN] . Your computer, of course, cannot only add, using +, but it can also subtract using –, multiply using*, divide using / and raise one number to the power of another using ↑. +, –, *, /, are called operations, and they operate on numbers called operands.

Example:

PRINT 3 ↑ 2 [RETURN]

and the answer 9 will appear.

ORDER OF NUMERIC OPERATIONS

When operations are combined, care must be taken to note the order in which the computer carries out the operations. The order is as follows:

1) Minus sign – used to indicate negative numbers.
2) Exponentiation starting at the left and moving right.
3) Multiplication and division (which are given the same order of precedence). Here too the computer moves from left to right.
4) Subtraction and addition moving from left to right.

A)

Example:

PRINT 3↑2↑2↑2 [RETURN] *6561*

This is done by squaring 3 to get 9. Then squaring 9 to get 81, and then squaring 81 to get 6561.

B)

Example:

*PRINT 6 * 2 + 3* [RETURN]

Here the computer first multiplies 6 X 2 and then adds 3.

C)

Example:

```
PRINT 6 + 3 * 4 + 6/3   [RETURN]
20
```

First the computer carries out the multiplication and division and then adds to give 6 + 12 + 2.

BRACKETS

All operations within brackets will be carried out first before the other operations.

Example:

```
PRINT 18/(3 + 3)   [RETURN]
3
```

Where brackets are placed within brackets the innermost brackets are calculated first.

Example:

```
PRINT 20/(1 + (3 ↑ 2))   [RETURN]
2
```

CHAPTER 4

CONSTANTS AND VARIABLES

- CONSTANTS
- VARIABLES
- LET
- SEMI-COLONS AND COMMAS
- COLONS

CONSTANTS

In the previous chapters we have been using constants. A constant is, of course, something which does not change and can be either positive or negative. The number 6.32 is a constant. Moving onto 6.33 just gives a different constant.

The range of a number in the computer is $-10^{38} \leq X \leq 10^{38}$.
The lowest positive number is 10^{-38}.

VARIABLES

A variable, as you might surmise, is something which changes. In Y = X + 3 both X and Y are variables as they have many possible values. A variable can be denoted by any letter of the alphabet, any two letters, or by a letter and a number providing the letter comes first. For example, A, AB, A6.

The variable name can be any length but only the first two characters will be recognised by the computer. For example, the computer will consider HELLO to be the same variable as HE.

A variable name cannot be or include any of the command words like **LET** or **PRINT**. there are 3 types of variables in total, namely:

a) real : — can have decimal value, e.g. A = 3.5, or A = 1E3.
b) integer : — cannot have decimal value, e.g. A% = 3.
c) string : — contain a string, e.g. A$ = "ABCD".

LET

The command **LET** can be used to assign a value to a variable. If a variable is not assigned a value it is assumed to be equal to zero. The variable will keep its assigned value until another **LET**, **READ** or an **INPUT** command is used to change the value.

Example:

```
LET A = 7    [RETURN]
LET B = 9    [RETURN]
PRINT A + B   [RETURN]
16
```

In BASIC the = sign does not mean the same as it usually does. Here it tells the computer to give the variable on the left hand side the same value as the right hand side.

The left hand side of the statement must always be a variable. Have a look at the next example.

Example:

```
LET A = 2    [RETURN]
LET B = 3    [RETURN]
LET C = 20   [RETURN]
LET A = 2 + A    [RETURN]
LET D = 3 + D    [RETURN]
PRINT A; B; C; D    [RETURN]
4  3  20  3
```


In the fourth line we see that A is assigned a new value of 2 plus the old value of A, which was also 2, giving a total of 4.

In the fifth line we see that D is given the value 3 + D, as zero is the value given to any variable without an assigned value. However, it is a good practice to predefine all variables.

Note that in your computer it is not strictly necessary to use **LET** to assign a value to variable, and *A = 7* will carry out the same function as *LET A = 7*.

SEMI-COLONS, COMMAS

If more than one item is included in a PRINT statement the items should be separated by either a (,) or (;). Note the use of the semi-colon in the PRINT statement on page 51. This causes the results to be printed immediately after each other with a space left for the sign of that number. When we use the semi-colon with strings there is no space.

A comma causes the result to be printed as follows. Think of your screen divided up into 2 sections of 16 characters. A comma will cause the first result to be printed at the beginning of the first section — on the left side — and the second result to be printed at the beginning of the second section. The third result will go back to the first section and will be printed under the first result. If a result is longer than 16 characters it will overlap into the next section. The next result will ignore this section and start at the beginning of the following one.

COLONS

If you have more than one statement on a line you must separate them by using colons.

Example:

```
10 FOR I = 1 TO 5 : PRINT I; : NEXT [RETURN]
RUN [RETURN]
1 2 3 4 5
```

LISTS OF PRINT STATEMENTS

Example:

```
10 PRINT 4  [RETURN]
20 PRINT 6  [RETURN]
30 PRINT 8  [RETURN]
RUN [RETURN]
```

Your screen will show

Example:

```
4
6
8
```

SEMI COLONS AT THE END OF PRINT STATEMENTS

Example:

```
1Ø PRINT 4;  RETURN
2Ø PRINT 5;  RETURN
3Ø PRINT 6;  RETURN
RUN  RETURN
```

Your screen will show

Example:

```
4  5  6
```

In general, a semicolon placed at the end of a **PRINT** statement tells the computer not to go to a new line after printing.

CHAPTER 5

PROGRAMMING

- REM
- INPUT
- NEW
- RUN
- LIST
- PAUSE IN LISTING
- DELETING A LINE

PROGRAMMING

OK, so now let's try some simple programming. In the last few chapters we have been dealing with "immediate execution" — with the computer obeying immediately. We now want the computer to store statements so that they can be executed later on — "deferred execution".

Have a look at this program.

Example:

```
10 REM RAISE TO THE POWER OF 3 [RETURN]
20 INPUT A  [RETURN]
30 PRINT A; A↑3  [RETURN]
```

Notice that each line begins with a number. These numbers tell the computer not to obey immediately but to store the lines away. The line number governs the order in which the line will appear on the screen. It is useful to write the numbers in tens as new lines can be later fitted into any part of the program by giving them a value of say 15 or 25. The range of possible line numbers is from 0 to 65529.

REM

The **REM** in line 10 is simply there to remind you later on of the purpose of the program. The computer will ignore any line which starts with REM. However **REM** lines use memory space so if you are short of space you can delete **REM** lines.

INPUT

The **INPUT** in line 20 asks you to assign a value to the variable A. When you run this program, a question mark "?" will be displayed. The computer will wait until you type in a value and give this value to the variable A.

NEW

So how do we feed the program into the computer? Well first feed in the NEW command and press RETURN. This will wipe out any old programs and variables. Remember the NEW command clears the memory of the computer.

Now type

Example:

```
10 REM RAISE TO THE POWER OF 3  RETURN
20 INPUT A  RETURN
30 PRINT A; A↑3  RETURN
```


You can now run the program by typing RUN and pressing RETURN . The sign '?' will now appear under RUN. This is the result of the **INPUT** statement and the computer is now waiting for you to give a value to the variable A. This value should be typed next to '?'.

Let's type 2 and RETURN

The screen will look like this.

Example:

```
10 REM RAISE TO THE POWER OF 3.  RETURN
20 INPUT A  RETURN
30 PRINT A;A↑3  RETURN
RUN  RETURN
?  2  RETURN
2  8
```

RUN

By typing [RUN] and pressing [RETURN] the whole of your stored program will be executed, starting at the line with the smallest line number. If however you press [RUN] (CTRL-6) and then a line number before [RETURN] , the program will be executed starting from that line.

Have a look at this program

Example:

```
10 INPUT A, B  [RETURN]
20 PRINT A + B  [RETURN]
RUN  [RETURN]
```

What will happen here is first of all you will get one '?' for you to put a value of A beside. On the next line you will get '??' for you to put the value of B beside. So carrying on with the program above.

Example:

```
10 INPUT A, B  [RETURN]
20 PRINT A + B  [RETURN]
RUN  [RETURN]
?    3  [RETURN]
??   6  [RETURN]
     9
```

One **INPUT** command will only accept up to two lines of variables. However, more than two values can be fed in via a single **INPUT** command, if commas are used to end each one instead of [RETURN] .

LIST

If you want the whole program to be displayed in an ascending line number order then just type **LIST** and press [RETURN] .

Example:

```
10 INPUT A  [RETURN]
20 INPUT B  [RETURN]
30 PRINT A; B; C; A + B + C  [RETURN]
25 INPUT C  [RETURN]
```

Example:

```
LIST  [RETURN]
10 INPUT A
20 INPUT B
25 INPUT C
30 PRINT A; B; C; A + B + C
```

If you only want one line to be displayed then type

Example:

```
LIST (Line number) [RETURN]
```

Example:

```
LIST 2Ø [RETURN]
2Ø INPUT B
```

To list part of a program say, lines 2Ø – 3Ø, type

Example:

```
LIST 2Ø – 3Ø [RETURN]
2Ø INPUT B
25 INPUT C
3Ø PRINT A; B; C; A + B + C
```

If you type *LIST – 3Ø* you will get the program listed up to line 3Ø from the start.

If you type *LIST 3Ø –* you will get the program listed from line 3Ø to the end.

PAUSE IN LISTING

If you have a very long program you might wish to have a look at a particular line while it is being listed. To do this just press the [SPACE] bar when you wish the listing program to stop.

Press the same bar again to continue.

DELETING A LINE

To get rid of any program line just type the line number and press [RETURN].

CHAPTER 6

COMPUTER PROGRAMMING REVISITED

- GOTO
- BREAK
- CONT
- STOP
- END
- CLEAR

Here are some more commands to help you write more interesting programs.

GOTO

This command tells the computer to go backwards or forwards to the line number following the GOTO statement and then to carry on executing the program from that line number.

Example:

```
10 INPUT A  [RETURN]
20 PRINT A, A↑3  [RETURN]
30 GOTO 10  [RETURN]
   RUN  [RETURN]
   ?
```

If you give the value say 2 to A the computer will return the results 2 and 8. However a question mark will again appear on the screen asking you to give A another value. This procedure is the result of the GOTO statement telling the computer not to end at line 30 but to go back to line 10 and start again.

$\boxed{\text{BREAK}}$

When you get tired of putting in different values of A you can press $\boxed{\text{BREAK}}$ (CTRL—). The computer will now stop executing the program and *BREAK IN 10* will appear on your screen. It should be noted that $\boxed{\text{BREAK}}$ is not a command. You can break any continuous program by pressing $\boxed{\text{BREAK}}$ (CTRL—).

CONT

If however, after stopping the program execution you feel there are still some values of A you would like to try you can type **CONT**, and the computer will start to execute the program once more. **CONT** causes a program to continue after stopping in response to $\boxed{\text{BREAK}}$ (CTRL- –) or STOP, without resetting any variables.

STOP

A useful statement in programming is **STOP**. This causes the program to stop at the line printed after the STOP statement and can help you to examine the results of the variables at various stages in the program. It is also extremely useful when it comes to locating mistakes (debugging). A liberal supply of STOP statements throughout a program is therefore a good idea, until you are sure that it is working properly.

You can restart the program by typing **CONT**. The program will carry on from the next line after the STOP.

END

The **END** statement is used to terminate execution. But unlike the **STOP** statement, execution cannot be continued after an **END** statement.

Example :

```
10  INPUT A [RETURN]
20  IF A>0 THEN PRINT "A IS POSITIVE":
    END [RETURN]
30  IF A<0 THEN PRINT "A IS NEGATIVE":
    END [RETURN]
40  PRINT "A IS ZERO" [RETURN]
50  END [RETURN]
```

NOTICE the **STOP** statement will give you the line number when it is executed. This will not happen with the **END** statement.

CLEAR

The **CLEAR** statement is used to assign more memory space for the string variables.

Example :

```
10 CLEAR 100 [RETURN]
```

This command will assign 100 bytes of memory for strings. If the **CLEAR** command is not used, the computer will assume the number of bytes of memory for strings to be 50. The use of the **CLEAR** command only will also reserve the same number of bytes. However, if a value follows the **CLEAR** command, the computer will assign the number of bytes of that value. If you want to use more strings in your program, set this number to a larger one but, at the same time, you will have less space for your program.

CHAPTER 7

NUMERIC FUNCTIONS

- ABS
- SGN
- SQR
- LOG
- EXP
- INT
- RND
- SIN
- COS
- TAN
- ATN

WHAT IS A FUNCTION?

A function is a 'law' which when applied to a certain value will give a new value. We call the first value the argument and the new value the result.

SQR is the square root function. So if we type

Example: *PRINT SQR (9)* [RETURN]

we will get the answer 3.

In this example 9 is the argument, SQR is the function and 3 is the result.

Below we give a list of the numeric functions and a brief explanation. Any function which we consider to be new to the reader will be explained in more detail afterwards. The functions will appear later on in programs so we don't give a sample program for each one here.

A LIST OF NUMERIC FUNCTIONS

Function	What it does
ABS (X)	Returns the absolute (positive) value of X
SGN (X)	Returns the sign of the argument X negative returns – 1 X positive returns + 1 X zero returns Ø
SQR (X)	Returns the square root of X. X cannot be negative.
LOG (X)	Gives the natural logarithm of X, i.e., the logarithm to the base e (=2.71828). The value of the argument must be greater than zero.
EXP (X)	Gives you the value e^X – i.e., the natural antilogarithm of X.
INT (X)	Gives the greatest integer which is less than or equal to X.
RND (X)	Gives a random whole number between 1 and X. If X equals zero, RND (X) returns a random number between Ø and 1. X cannot be negative.
SIN (X) COS (X) TAN (X)	The argument of the trigonometrical functions is taken to be in radians (1 radian = 36Ø/2π = 57.296 degrees). The range of X is –9999999 ≤ (X) ≤ 9999999.
ATN (X)	This gives the result of ARC TANGENT in radians.

A FURTHER LOOK AT ABS : SGN : INT : RND

ABS (X)

This gives the absolute (positive) value of the argument. So *ABS (–7) = 7.*

Example:

*PRINT ABS (7 – 2 * 4)* [RETURN]
1

SGN (X)

This function will give the value of +1 if X is positive, Ø if X is zero, and –1 if X is negative. So *SGN (4.3) = 1; SGN (Ø) = Ø; SGN (–.276) = –1.*

Example:

A = –6 [RETURN]
PRINT SGN (A); SGN (A – A) [RETURN]
–1 Ø

INT (X)

This converts arguments which are not whole into the largest whole number below the argument. So *INT (5.9) = 5;* also *INT (–5.9) = –6.* Note that with negative arguments, the *absolute value* of the result returned by INT will be greater than that of the argument.

Example:

PRINT INT (-6.7) [RETURN]
–7

RND (X)

This will produce a random number between 1 and X if X is positive.

Example:

PRINT RND (19) [RETURN]

You will get a number between 1 and 19. *RND (Ø)* will give you a number between Ø and 1.

Note: X cannot be negative.

CHAPTER 8

STRINGS

- STRING VARIABLES
- STRING FUNCTIONS
- LEN
- STR$
- VAL
- LEFT$
- RIGHT$
- MID$
- ASC
- CHR$
- STRING COMPARISONS
- INKEY$

STRINGS

Note: We assume that you are now familiar with the use of the RETURN key so we will not keep reminding you of it.

A string is any combination of CHARACTERS that is treated as a unit.

String constants must be enclosed in inverted COMMAS.

Example:

```
"HELP"
```

When using the **PRINT** statement a semi-colon between strings will not cause a space to appear between the results. They will appear immediately next to each other.

STRING VARIABLES

Any letter of the alphabet or letter followed by a number digit can be used as a string variable but must be followed by a $ sign. The computer accepts these characters as the variant name.

Example:

```
A$ = "ONE DOZEN EGGS"
```

You can add strings to each other. The is called concatenation. You cannot subtract, divide or multiply strings.

Example:

```
10 A$ = "I A"
20 B$ = "M 15 YEA"
30 C$ = "RS OLD"
40 PRINT A$ + B$ + C$
   RUN
   I AM 15 YEARS OLD
```

Notice the spacing of the string characters here.

STRING FUNCTIONS

We can also use functions to act on strings. Have a look at the following:

LEN

This function works out the length of the string argument, which must be in brackets. So if you type *PRINT LEN ("JOHN")* the computer will return the result 4. This is telling you that there are 4 characters in the string "JOHN". Blank spaces have the value of a character. Thus if you put in spaces "J O H N" it comes out as 7 characters.

STR$

The STR$ function changes a number argument into a string. Let us take a look at the following example and see how it works.

Example:

```
A$ = STR$ (73)
```

This is the same as saying

Example:

```
A$ = "73"
```

Here is an example program

Example:

```
10 A$ = STR$ (7 * 3)
20 B$ = A$ + "BIG"
30 PRINT B$
   RUN
   21BIG
```

VAL

VAL works like STR$ but in reverse. It changes a string argument into a number. It only works on numbers not on operators or other characters.

Look at the following short program

Example:

```
10 A$ = "33"
20 B$ = "20"
30 C = VAL (A$ + B$)
40 PRINT C;C + 100
RUN
3320    3420
```

SUBSTRINGS

It is also possible to get substrings of strings. A substring is as you might guess a part of a string. For example: "ABC" is a substring of "ABCDE".

LEFT$ (A$,N)

This will return the substring from the leftmost of string A$ — the first character — to the Nth character.

Example:

```
10 A$ = "ABCDE"
20 B$ = LEFT$ (A$ + "FGH", 6)
30 PRINT B$
   RUN
   ABCDEF
```

RIGHT$ (A$,N)

This will return a substring as in the above example but starting from the Nth character from the end and running to the last one — the right most character in the string A$.

Example:

```
10 A$ = "WHY"
20 B$ = RIGHT$ (A$ + "ME", 4)
30 PRINT B$
   RUN
   HYME
```

MID$ (A$, M, N)

This function returns a substring of the string A$ starting from the Mth character with a length of N characters.

Example:

```
10 A$ = "ABCDEFGH"
20 B$ = MID$ (A$, 2, 3)
30 PRINT B$
   RUN
   BCD
```

ASC (A$)

The ASC statement which is written as **ASC (A$)** where **A$** is a variable string expression, will return the ASCII code (in decimal) for the FIRST character of the specified string. Brackets must enclose the string specified. Refer to the appendix for the ASCII code. For example the ASCII decimal value of "X" is 88. If *A$ = "XAB"*, then *ASC (B$) = 88*.

Example:

```
10 X = ASC ("ROY")
20 PRINT X
   RUN
   82
```


CHR$ (N)

This statement works the opposite way around to the **ASC** statement. The **CHR$** statement will return the string character which corresponds to the given ASCII code. The augument may be any number from Ø to 255 or any variable expression with a value within that range. Brackets must be put around the argument.

Example:

```
30 PRINT CHR$ (68)
   RUN
   D
```

STRING COMPARISONS

Relational operators can be applied to string expressions to compare the strings for equality or alphabetic precedence. As far as equality is concerned all the characters (and any blanks) must be identical and in the same order.

Example:

```
10 A$ = "AA"
20 B$ = "BA"
30 IF A$ = B$ then PRINT 20
40 IF A$ < B$ then PRINT 30
50 IF A$ >B$ then PRINT 40
   RUN
   30
```


The comparisons are done by taking the ASCII value of the string characters from the table in the appendix and then comparing these values. The table gives us the value for 'A' as 65 and 'B' as 66. The program above is therefore asking for confirmation that 65 is less than 66.

If the first two CHARACTERS of a string are equal the computer will search for the third CHARACTER and do the comparison on this.

Example:

```
10 A$ = "ABC"
20 B$ = "ABD"
30 IF B$ >A$ then PRINT 40
   RUN
   40
```

Here the critical comparison is between the characters C and D. The ASCII table value of C is 67 and the table value of D is 68. B$ is therefore greater than A$.

INKEY$

INKEY$ returns either a one-character string containing a character read from the keyboard or a null string (i.e., an empty string, with no characters) if no key has been pressed at the keyboard. All characters are passed through to the program except for $\boxed{\text{BREAK}}$ (CTRL- —), which terminates the program.

Example:

```
10 A$ = INKEY$
20 PRINT A$;
30 GOTO 10
```

To stop this program, press $\boxed{\text{BREAK}}$ (CTRL- —).

CHAPTER 9

CONDITIONS

- IF . . . THEN . . . ELSE
- CONDITIONAL BRANCHING
- LOGICAL OPERATORS
- TRUTH TABLES

IF . . . THEN . . . ELSE

As we make our way through BASIC, we find that we gain more control over the computer, that is, we are able to do more with the computer. In this chapter we are going to take a look at the "**IF . . . THEN . . . ELSE**" statement. This is, perhaps, one of the two most important programming concepts in BASIC. The other one is "**FOR . . . NEXT**". We will look at this in the following chapter.

Let us look at this example.

Example:

```
60 IF A$ >B$ THEN PRINT A$ ELSE PRINT B$
```

This tells the computer that if the expression A$ is greater than B$ to carry out the statement *PRINT A$;* otherwise it should carry out the statement *PRINT B$.*

CONDITIONAL BRANCHING

In general terms , the **IF . . . THEN . . .** statement is used for conditional branching. It uses the general form "**IF** (condition) **THEN** (action clause)." A condition is made up of an expression, a relation and an expression.

Any BASIC expressions may be used but both expressions must be of the same type, that is either both numeric or both string expressions.

Relations or comparisons used in the **IF . . . THEN** statement are the following:

= Equal to
<= Less than or equal to
< > Not equal to
> = Greater than or equal to
< Less than
> Greater than

Here are some more examples of how we can use conditionals.

IF . . . . THEN A = B
IF . . . . THEN GOTO
IF . . . . THEN GOSUB
IF . . . . THEN PRINT
IF . . . . THEN INPUT

Example:

```
3Ø IF X>25 THEN 6Ø
```

Here if the condition X > 25 is true, the computer is told to jump to line 6Ø (Note: the **GOTO** is optional after **THEN**).

If the condition is not true, that is, if X is not greater than 25 then the computer simply carries on with the normal line number order in the program. Notice that it is not necessary to use the **ELSE** part of the command here as this is optional.

Example:

```
1Ø INPUT A, B
2Ø IF A>B THEN 5Ø
3Ø IF A<B THEN 6Ø
4Ø IF A = B THEN 7Ø
5Ø PRINT A; "IS GREATER THAN"; B: END
6Ø PRINT A; "IS LESS THAN"; B: END
7Ø PRINT A; "IS EQUAL TO "; B
8Ø END
   RUN
   ?       7
  ??       3
   7 IS GREATER THAN 3
```

Example:

```
4Ø IF P = 6 THEN PRINT "TRUE" ELSE PRINT
"FALSE"
```

In this example if P = 6 the computer will print TRUE. Any other value will produce a FALSE. In either case the computer will carry onto the next line.

It is possible for more than one statement to follow the **THEN** or **ELSE** command.

Example:

```
50 IF A = 5 THEN PRINT "TRUE": S = S – 3:
GO TO 90 ELSE PRINT "FALSE": K = K + 8
```

So if A equals 5 the computer will print TRUE, subtract 3 from the variable S and go to line 90. If A does not equal 5 the computer will print FALSE, add 8 to the variable K and then carry on with the next normal line.

LOGICAL OPERATORS

Logical operators are used in IF . . . THEN . . . ELSE and such statements where a condition is used to determine subsequent operations within the user program. The logical operators are: AND, OR, NOT.

For purposes of this discussion A and B are relational expressions having only TRUE and FALSE. Logical operations are performed after arithmetical and relational operations.

Operator	Example	Meaning
NOT	**NOT A**	If A is true, NOT A is false.
AND	**A AND B**	A AND B has the value true, only if A and B are both true. A AND B has the value false if either A or B is false.
OR	**A OR B**	A OR B has the value true if either A or B or both are true. It has the value false if both are false.

TRUTH TABLES

The following tables are called TRUTH TABLES. They illustrate the results of the above logical operations with both A and B given for every possible combination of values.

TRUTH TABLE FOR "NOT" FUNCTION

A	NOT A
T	F
F	T

TRUTH TABLE FOR "AND" FUNCTION

A	B	A AND B
T	T	T
T	F	F
F	T	F
F	F	F

Note that T = TRUE and F = FALSE.

TRUTH TABLE FOR "OR" FUNCTION

A	B	A OR B
T	T	T
T	F	T
F	T	T
F	F	F

NOTE: T = TRUE and F = FALSE

Example:

```
10 INPUT A, B, C
20 IF A = B AND B = C THEN PRINT "A = B = C"
30 IF (NOT A = B) OR (NOT B = C) THEN 50
40 END
50 PRINT "A = B = C IS FALSE"
60END
   RUN
   ?   10
   ??   5
   ??   7
   A = B = C IS FALSE
```

Moreover AND, OR, NOT can be used to manipulate numerical values. These operations are based on binary numbers with 1 and Ø representing TRUE and FALSE respectively. For example:

i) NOT 1 = -2 [1 = binary ØØØØØØØ1 and −2 = binary 1111111Ø, so it just changes the 1 to Ø and Ø to 1. In other words, TRUE (1) changed to FALSE (Ø) and FALSE (Ø) is changed to TRUE (1).]

ii) 6 OR 13 = 15 [6 = binary ØØØØØ1 1 Ø and 13 = binary Ø Ø Ø Ø 11 Ø 1, so with reference to the OR truth table, 6 OR 13 = 15 = binary Ø Ø Ø Ø1111]

iii) 6 AND 13 = 4 [6 = binary ØØØØØ1 1 Øand 13 = binary Ø Ø Ø Ø 1 1 Ø 1, so with reference to the AND truth table, 6 AND 13 = 4 binary ØØØØØ1 ØØ]

CHAPTER 10

LOOPING

- FOR . . . TO
- NEXT
- STEP

FOR . . . TO . . . NEXT . . . STEP

Often we need the computer to perform repetitive tasks. The looping process enables us to do just this without having to type in similar statements many times.

For example if we want many numbers cubed and then divided by 3 we don't have to type in various values; all we have to do is something like this:

Example:

```
10 FOR X = 1 TO 10
20 PRINT X; (X↑3)/3
30 NEXT X
40 END
   RUN
    1   .333333
    2   2.66667
    3   9.00001
    4   21.3333
    5   41.6667
    6   72
    7   114.333
    8   170.667
    9   243
    10   333.333
```


From the above example we can see that the computer has cubed and divided by 3 the numbers between 1 and 10. The FOR . . . TO . . . statement, therefore, stipulates the range of numbers you wish to act on.

You will notice that the number were incremented (increased) by one each time. The increment size can be changed by using the STEP statement and a positive number. If a negative number follows the STEP statement we will get a decrement (decrease). It is also possible to use a decimal, an expression or a variable.

Example:

```
10 FOR X = 1 TO 10STEP 2
20 PRINT X; (X↑3)/3
30 NEXT X
40 END
   RUN
```

This will act on all the odd numbers between 1 and 10and your screen will show the following numbers:

Example:

```
1  .333333
3  9.000001
5  41.6667
7  114.333
9  243
```

You will also notice that each **FOR** loop must be closed with a **NEXT** statement. The variable name of the **NEXT** statement must be the same as the variable name of the **FOR** statement – in this case X. On your computer the variable name following the **NEXT** statement can be omitted if you wish. However it is good programming practice to put it in. This can avoid unexpected results (and errors) when you have a number of **FOR-NEXT** loops – particularly if they are "nested" (explained shortly).

Loops are very useful for writing tables as you will see from the following example.

Example:

```
10 REM TO PRINT A SINE AND COSINE TABLE
20 PRINT "SIN (X)", "COS (X)"
30 FOR X = 0 TO 2 STEP 0.5
40 PRINT SIN (X), COS(X)
50 NEXT X
60 END
   RUN

SIN (X)                    COS (X)
0                          1
.479426                    .877582
.841471                    .540302
.997495                    .0707371
.909298                    -.416147
```

If you do use a variable name following your **NEXT** statement and you use 2 loops you must be careful not to cross your loops.

Example:

```
10. . . . . . . . . . .
20 FOR X = 0 TO 10
30. . . . . . . . . . . . . . .
40 FOR Y = 0 TO 5
50. . . . . . . . . .
60 NEXT X
70 NEXT Y
```

This causes crossed loops and will result in a NEXT WITHOUT FOR error message. The correct way is as follows:

```
10. . . . . . . . . . . . . .
20 FOR X = 0 TO 10
30 . . . . . . . . . . . .
40 FOR Y = 0 TO 5
50. . . . . . . . . . . . . .
60 NEXT Y
70 NEXT X
```


This gives you the loop for Y "nested" or fully inside the loop for X.

With your computer, the number of levels of nesting for **FOR-NEXT** loops depends on the memory size of the computer.

CHAPTER 11

SUBROUTINES

- GOSUB
- RETURN

GOSUB – RETURN

A program has a beginning and an end. It has a structure. This structure is made up of smaller building blocks. You may need some of these blocks or sections of the program many times in various places in the overall program. To help us deal with these similar smaller parts of the program we can use subroutines. The statements we use are **GOSUB** and **RETURN.**

```
10 PRINT "HELLO"
20 GOSUB 50
30 PRINT "GOODBYE"
40 END
50 PRINT "HOW ARE YOU?"
60 GOSUB 80
70 RETURN
80 PRINT "SEE YOU"
90 RETURN
   RUN
   HELLO
   HOW ARE YOU?
   SEE YOU
   GOODBYE
```


The lines are executed in this order 10, 20, 50, 60, 80, 90, 70, 30, 40. Hopefully you can see from this example how the GOSUB statement works.

The **GOSUB** statement tells the computer to move on to the line number indicated, that is to the line number following **GOSUB.** However unlike the **GOTO** command the **GOSUB** command makes the computer "remember" where it is jumping from, so it can come back again to the statement that immediately follows the **GOSUB** statement.

The computer will carry on with the subroutine until the **RETURN** statement is met. It is the **RETURN** statement that makes the computer go back to the statement following the **GOSUB** statement.

For another example:

20 GOSUB 60 tells the computer to jump to line 60. The computer will start executing at line 60 until it meets a **RETURN** statement. On meeting the **RETURN** statement the computer will go back and start executing at the statement following the **GOSUB** statement in line 20.

Have a look at this example and see if you can work out what's happening.

```
10 FOR X = 1 TO 5
20 GOSUB 60
30 PRINT X; S
40 NEXT X
50 END
60 S = 0
70 FOR J = 1 TO X
80 S = S + J
90 NEXT J
100 RETURN
   RUN

   1     1
   2     3
   3     6
   4     10
   5     15
```

Get the idea? Here the subroutine is the section from lines 60 to **100** inclusive, and we are "calling it" (i.e., using it) a total of 5 times.

CHAPTER 12

LISTS AND TABLES

- ARRAYS
- DIM

ARRAYS AND DIM

There are two types of variables — Simple Variables and Array Variables. Up to now we have been dealing with simple variables. Let us now take a look at the array type.

An array is an organised list of values which provides an efficient way of handling large amounts of information. The values can be either numbers or strings. To set up an array you first have to give the array a name and a size. The name can be either a letter or a string e.g. A$ (5).

It is easy to distinguish an array variable from a simple variable. The array variable is always followed by brackets containing a number. e.g. A(2), B(7), G5$(7). This number in the brackets is called a subscript.

WHY USE ARRAYS?

Let us suppose you have a number of books at home — say 10 0 books — and you want to index all your books. If we assign a variable to each book name for example:

Example:

```
10 A1$ = "GONE WITH THE WIND"
20 A2$ = "OLIVER TWIST"
   .
   .
   .
   .
   .
   .
1000L1$ = "BASIC PROGRAMMING"
```

This would be very inefficient and time consuming. A better way of dealing with this list is to use an ARRAY. The variable A$ will stand for the list of books. Let us look at the following example:

Example:

```
10 REM BOOK NAMES
20 DIM A$(99)
30 FOR X = 0 TO 99
40 INPUT "BOOK NAME"; A$ (X)
50 NEXT X
   RUN
   BOOK NAME?
```

After you give a name after to the question mark, 'BOOK NAME?' will appear again underneath. This will carry on until your list is completed.

Before you can use an ARRAY it is necessary to use the DIM statement. Above we have DIM A$ (99). This tells the computer to reserve space for the array called A$ and that array has 100 subscript variables (from A$ (0) to A$ (99) inclusive). It is possible at a later time to sort, rearrange or print out this set of data. This type of array is called an one dimensional array and it deals with lists. DIM stands for dimension.

It is also possible to have a two dimensional array where we have two subscripts and we are dealing with numbers in the matrix form.

Let us suppose we have 5 students doing 3 exams. The results of the exams look like this

	EXAM (1)	EXAM (2)	EXAM (3)
STUDENT 1	50%	70%	90%
STUDENT 2	63	42	36
STUDENT 3	20	62	50
STUDENT 4	70	75	84
STUDENT 5	93	82	68

These results can be recorded on the computer using a two dimensional array. We would have to start with the statement: *DIM A (4,2)*. Here 4 is one less than the number of students and 2 is one less than the number of columns and in this case exams. So A(3, 0) will be 70. This is the score of the fourth student in the first exam.

It is possible to have up to a three dimensional ARRAY – DIM A(3, 6, 2). The size of each dimension is limited by the memory size of the computer so remember; A(X) is an one dimensional array variable, A(X, Y) is a two dimensional array variable, and A(X, Y, Z) is a three dimensional one.

Note that if you do not use the DIM statement the subscripts 0 – 10 are allowed for each dimension of each array used by default.

CHAPTER 13

READ, DATA, RESTORE

- READ
- DATA
- RESTORE

READ AND DATA

When it is necessary to enter a lot of information or data into the computer, using the **INPUT** statement can be very time consuming. To help us out here we can use the **READ** and **DATA** commands.

Example:

```
10 DATA 10, 60, 70, 80, 90
20 READ A, B, C, D, E
30 PRINT A; B; C; D; E
   RUN
   10   60   70   80   90
```

The **READ** statement consists of a list of variable names with commas between each variable.

The **DATA** statement consists of a list of expressions separated by commas. These expressions can be either numeric or strings. The **READ** statement makes the computer look up the value of its variables from the **DATA** statement. When the computer goes to **READ** first it will assign the first expression from the **DATA** list. The next time it goes to **READ** it will assign the second value – and so on. If the **READ** runs out of **DATA** you will get '? **OUT OF DATA ERROR**'.

RESTORE

If you want to use the same data later on in the program you can do so by using the **RESTORE** statement.

Example:

```
10 DATA 1, 3, 6, 9
20 READ A, B, D
30 RESTORE
40 READ X, Y
50 PRINT A; B
60 PRINT X; Y
70 END
   RUN
    1    3
    1    3
```

The **RESTORE** command makes subsequent **READ** statement get their values form the start of the first **DATA** statement.

Now see if you can work out what is happening here.

Example:

```
10 REM FIND AVERAGE
20 DATA 0. 125, 3, 0.6, 7
30 DATA 23, 9.3, 25.2, 8
40 S = 0
50 FOR I = 1 TO 8
60 READ N
70 S = S +N
80 NEXT
90 A = S/8
100 PRINT A
    RUN
    9.52813
```

Now using our student's examinations results form the chapter on arrays (chapter 12) see how the **READ** and **DATA** commands can be used.

Example:

```
10 CLS: DIM A(4, 2)
20 PRINT "RESULT": PRINT
30 PRINT TAB(8); "EX(1)  EX(2)  EX(3)"
40 PRINT
50 FOR J = 0TO 4
60 PRINT "STUDENT"; J + 1;
70 FOR I = 0TO 2: READ A(J, I):PRINT A(J,I);:
   NEXT: PRINT
80 NEXT
90 END
100 DATA 50, 70, 90, 63, 42, 36 20, 62, 50, 70, 75
110 DATA 84, 93, 82, 68
   RUN
```

CHAPTER 14

PEEK AND POKE

- PEEK
- POKE

PEEK (address)

The PEEK function will return the value stored at the specified address in the memory of the computer. The value will be displayed in the form of a decimal number whose value is in the range of Ø – 255.

Example:

```
3Ø A = PEEK (28672)
```

This returns the value the program has at 28672 and gives this value to A. It should be noted that the address needs not be a value; it can be an expression.

POKE address value, expression

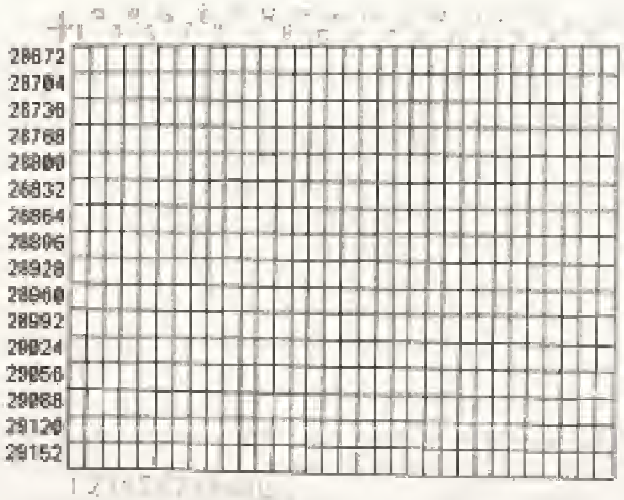
The POKE function complements the PEEK function. It sends a value to the stated address location. You need therefore an address and value, and again the value has to be between Ø and 255.

Example:

```
1Ø A = 1
2Ø POKE 29ØØØ, A
3Ø B = PEEK (29ØØØ)
4Ø PRINT B
   RUN
   1
```

When using this command you must be very careful as it can destroy your program. It is wise to save the program before you execute POKE. It is not recommended for newcomers without prior knowledge of what it does. You can only POKE to the Random Access Memory (RAM), that is to the place where the computer stores the information it wants to keep like your BASIC program, variables, the picture for the television and the various musical notes. So maybe the address you indicate may not be in the memory of the computer. The possible address is in the range of –32768 to +32767. (but not all of this range is occupied by RAM).

Screen RAM location for **POKE** and **PEEK** commands directly to the screen are Hexadecimal 7000–71FF (Decimal 28672–29183) in TEXT mode and 7000–77FF (Decimal 28672–30719) in GRAPHICS mode. (graphics mode is explained later). Starting location is decimal 28672.



Example:

```
10 CLS : SC = 28672
20 FOR I = 1 TO 9
30 READ A
40 POKE SC + I * 32, A
50 NEXT
60 GOTO 60
70 DATA 80, 79, 75, 69, 32, 80, 69, 69, 75
```

This example **POKEs** out inverse characters to TV screen. Press [BREAK] (CTRL- →) to stop this program.

Example:

```
10 CLS: SC = 28672
20 PRINT "PEEK" : PRINT
30 FOR I = 0 TO 3
40 PRINT PEEK (SC + I);
50 NEXT
```

This example **PEEKs** back the characters on TV screen.

Your computer has some different character codes to most other computers. They are shown below:

CHARACTER CODE (FOR POKE & PEEK)

(A)		(A)		(A)		(A)	
Ø	@	16	P	32		48	Ø
1	A	17	Q	33	!	49	1
2	B	18	R	34	"	5Ø	2
3	C	19	S	35	#	51	3
4	D	2Ø	T	36	$	52	4
5	E	21	U	37	%	53	5
6	F	22	V	38	&	54	6
7	G	23	W	39	'	55	7
8	H	24	X	4Ø	(	56	8
9	I	25	Y	41	)	57	9
1Ø	J	26	Z	42	*	58	:
11	K	27	[	43	+	59	;
12	L	28	\	44	,	6Ø	<
13	M	29	]	45	–	61	=
14	N	3Ø	↑	46	.	62	>
15	O	31	←	47	/	63	?

(A) 1ST COLUMN – CHARACTER CODE

2ND COLUMN – CHARACTER REPRESENTED

Note: These codes are for use with the POKE and PEEK commands. For PRINT CHR$(N), use the normal ASCII codes given in the table in the Appendix.

For A = Ø – 63

gives normal characters.

By adding an offset
A = Ø – 63 (+64) = 64 – 127
gives inverse characters.

(A)		(A)	
128		136	
129		137	
13Ø		138	
131		139	
132		14Ø	
133		141	
134		142	
135		143	

For A = 128 – 255 the codes are divided into 8 (graphic characters) groups each with different color code. (The shaded area has color.)

For A = 128 – 143 GREEN
= 144 – 159 YELLOW
= 16Ø – 175 BLUE
= 176 – 191 RED
= 192 – 2Ø7 BUFF
= 2Ø8 – 223 CYAN
= 224 – 239 MAGENTA
= 24Ø – 255 ORANGE

The whole table is also whown in the Appendix.

CHAPTER 15

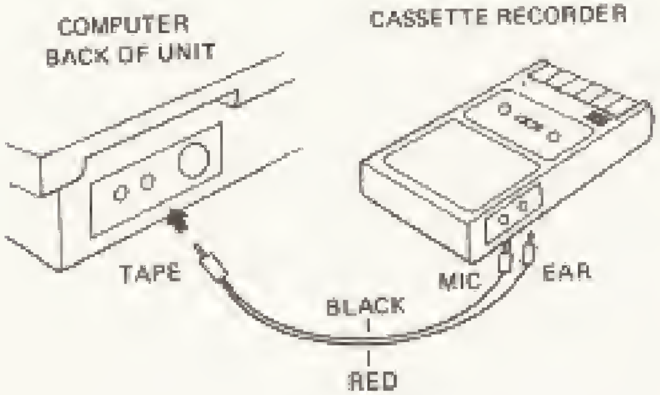
STORING PROGRAMS ON TAPE

- SETTING IT UP
- CLOAD
- CSAVE
- VERIFY
- CRUN
- PRINT #
- INPUT #

You may have developed some programs which you want to retain. It is too much trouble to type in a program, especially if it is long, every time you want to use it. This problem is easily solved. You can store your programs on tape and whenever you need them, load them into memory.

SETTING IT UP

To do this you need an ordinary cassette tape recorder, a cassette tape and interconnecting cords. Connect the recorder as shown in the picture.



You need to be familiar with three commands, namely, CSAVE, CLOAD and VERIFY.

You have received a cassette tape with your computer. On one side of this tape there is a program and the other side is blank. It is suggested that you start by loading this program into the computer.

There is a file name for each program on the tape. A file name is a "must" for saving a program but not absolutely necessary for loading and verifying a program.

The file name can be one to sixteen characters in length. The first character must be a letter; the rest can be any character. For our purpose saving a program means transferring a program that you have typed into the computer, to a tape.

Verifying a program means checking to make sure that the program on the tape is the same as the program in the computer.

Loading a program means transferring a program from the tape to the computer.

CLOAD "FILE NAME"

The procedure for loading a program from the tape to the computer is as follows:

1. Load the tape containing the required program into the recorder.

2. Rewind the tape to the start of the required program.

3. Type the COMMAND *CLOAD "FILE NAME".*

   BE SURE NOT TO PRESS THE [RETURN] KEY

4. Press the play button on the recorder.

5. Press the [RETURN] key.

6. If the computer finds no program on the tape, the statement **WAITING** is displayed on the screen. If you want the computer to come out of WAITING, PRESS [BREAK] CTRL- —) before stopping the cassette recorder.

7. If the incoming program has the file name which does not match with the specified one, then the statement *'FOUND T: FILE NAME'* appears and the program is skipped.

8. The desired program is loaded with the statement *'LOADING T: FILE NAME'* appears.

9. When the statement READY is displayed, press the STOP button on the recorder.

Let us suppose there are three programs on the tape and you have given them file names: PROGRAM 1, PROGRAM 2, PROGRAM 3. You want PROGRAM3 and it is at the end of the tape. You can type: *CLOAD "PROGRAM 3".* Then your screen will show:

Example:

```
CLOAD "PROGRAM 3"
WAITING
FOUND T: PROGRAM 1
FOUND T: PROGRAM 2
LOADING T: PROGRAM 3
READY
```

If you know the location of PROGRAM 3 and you set the tape at the beginning of this program the screen will show:

Example:

```
CLOAD "PROGRAM 3"
WAITING
LOADING: PROGRAM 3
READY
```

NOTE T: stands for TEXT file.

CSAVE "FILE NAME"

If you wish to save a program, make sure that you use a good quality tape. The quality of the tape can affect the quality of your recording.

It is important to set the volume of the cassette recorder within a proper range. This range will vary from one recorder to another. The tone should be set to MAXIMUM level.

The procedure for storing/saving a program is as follows:

1. Type in the complete program. It is advisable to use a short program at the start. Longer programs can be saved once you have achieved success with a short program.

2. Type in the COMMAND *CSAVE "FILE NAME"*

   Remember a File Name is a "MUST" for saving a program.

   BE SURE NOT TO PRESS THE $\boxed{\text{RETURN}}$ KEY

3. Load the recorder with a good quality tape.

4. Press the PLAY and RECORD buttons on the recorder.

5. Press the $\boxed{\text{RETURN}}$ key.

   The flashing CURSOR will disappear and the storing begins.

6. When the flashing CURSOR reappears, the storing is completed.

7. Press the STOP button on the recorder.

   The program that you typed in is now stored on a tape. To make sure that it is stored the user may verify this for himself.

VERIFY "FILE NAME"

To verify a program on tape (usually just after you have CSAVE'd it). The procedure is as follows:

1. List the program in the computer to make sure that the program still exists.

2. Type the COMMAND *VERIFY "FILE NAME"*

   BE SURE NOT TO PRESS THE RETURN KEY.

3. Press the PLAY button on the recorder.

4. Press the RETURN key. The flashing CURSOR will disappear and the verifying begins.

Example:

```
VERIFY "PROGRAM 2"
WAITING
FOUND T: PROGRAM 1
LOADING T: PROGRAM 2
VERIFY OK
READY
```


5. The 'OK' statement tells us that the programs on the tape is the same as the program in the computer.

6. If the verifying reveals an error, the statement 'VERIFY ERROR' will be displayed on the screen. This statement shows that the program on the tape is different from the one in the computer. In this case the user should CSAVE the program and verify it once again.

You can verify that there is a program on the tape by listening. If there is a program on the tape and your run it on the cassette recorder, the recorder will give out a distinctive sound.

CRUN "FILE NAME"

One more powerful command 'CRUN' can be used. This command is similar to 'CLOAD' except that the loaded program will start execution automatically after the loading is completed.

The four cassette interface commands, CSAVE, VERIFY, CLOAD and CRUN help the user to save his programs, verify them, and get them back into the computer to execute. The user should pay attention to the volume level of the recorder. Cassette interface is a means of inexpensive MASS STORAGE.

There are two more cassette type commands that the user should become familiar with. They are:

PRINT

PRINT# "FILE NAME", item list.
Sends the values of the specified variables or data onto a cassette tape. It is understood that the recorder must be properly connected and set in record mode when this statement is executed.

INPUT

INPUT # "FILE NAME", item list.
Inputs the specified number of values stored on cassette and assigns them to the specified variable names.

Example:

```
10 PRINT # "KAM", 1, 2, 3, 4, 5
   RUN
```

The data constants 1 to 5 are saved in the data file "KAM". BE SURE to put your cassette recorder in RECORD mode BEFORE execution.

Example:

```
10 INPUT # "KAM" , A, B, C, D, E
20 PRINT A; B; C; D; E;
   RUN
   FOUND D: KAM
   1  2  3  4  5
```

The data in the data file "KAM" are assigned to the variables A to E. BE SURE to put your cassette recorder in PLAY mode BEFORE execution.

NOTE D: stands for DATA file.

NOTE: If you are still unsuccessful in trying to **CSAVE**, **CLOAD**, **CRUN** or **VERIFY** your tape programs after several attempts, you may try to use another brand or model of cassette recorder to test the result. This is because not all of the recorders are suitable for data recording.

CHAPTER 16

GRAPHICS

- GRAPHICS MODES
- GRAPHICS CHARACTERS
- INVERSE
- SET
- RESET
- POINT

GRAPHICS MODES

There are 2 different modes of screen display. Once the power is turned on, the computer is in the normal Text mode, MODE (Ø), with 32 characters multiplied by 16 lines. This mode also dislplays 64 x 32 Dots with 9 colors. It is best used when some low resolution graphics and text are required, as the video output is more attractive and appealing.

The user can switch to the Higher Resolution Graphics mode, which has 128 x 64 Dots and 8 colors by making use of the MODE (1) command. This mode provides extra-fine graphics and is very suitable for games purposes.

To get back to normal Text mode you should use the **MODE** (Ø) command.

GRAPHICS CHARACTERS

There are 16 built-in graphics characters in your computer which can be typed by pressing the [SHIFT] key and any of the corresponding keys. These characters are useful for drawing pictures and graphs.

Have a look at the program beneath to see how you can use these characters.

Example:

```
1Ø REM COLOUR
2Ø FOR I = 1 TO 52
3Ø FOR J = 1 TO 8
4Ø COLOR J
5Ø PRINT "■";
6Ø NEXT J
7Ø NEXT I
8Ø GO TO 8Ø
```

Press [BREAK] (CTRL- —) to stop this program.

This example uses the graphics character shown in the program to plot some graphics. This example also shows all the 8 different colours in the TEXT mode by using graphic character.

This example only uses one graphic character. For the rest of them, you may observe your keyboard to see where they are located.

INVERSE

The inverse characters can be produced by simply pressing INVERSE (CTRL- :). Your computer will remain in the inverse mode until the INVERSE (CTRL- :) or RETURN key is pressed.

MODE (1) GRAPHICS

The following commands SET, RESET and POINT may be used for plotting graphics in MODE (1).

SET (X, Y)

On your computer this is used when dealing with colours. It plots a dot at a specified location on the screen which is determined by the values of X and Y. The value of X can be from Ø to 127 and the value of Y can be from Ø to 63.

RESET (X, Y)

This is used to wiped out a point switched on by the SET command. The X and Y values determine the location of the point to be wiped out. This is done by making the point the same colour as the background.

POINT (X, Y)

This tells you if a specific point has been plotted by the SET command. If the specified point has been plotted, POINT (X, Y) will return its colour code. Usually it is used with the IF-THEN-ELSE statement.

Like this:

Example:

```
8Ø SET (4Ø, 2Ø) : IF POINT (4Ø, 2Ø) THEN
   PRINT "YES" ELSE PRINT "NO"
```

Here is another example of the use of SET, RESET and also one using POINT.

Example of **SET** and **RESET**:

Example:

```
10 MODE (1) : COLOR 2
20 FOR I = 0  TO 127
30 SET (I, I/2)
40 NEXT
50 FOR J = 1 TO 1000: NEXT
60 FOR I = 0 TO 127
70 RESET (I, I/2)
80 NEXT
90 FOR J = 1 TO 1000: NEXT
100 MODE (0)
```

This example will plot a diagonal line on the screen and then rub this line out. When this program is completed, the computer will return back to **MODE (0)**, that is TEXT mode.

NOTE: If you want to keep the computer in graphics mode, replace line **100** by: *100 GOTO 100*, and press BREAK (CTRL- –) to stop this program.

Example of **POINT**:

```
10 MODE (1) : DIM A(4)
20 FOR I = 1 TO 4 : FOR J = 0 TO 127
30 COLOR I
40 SET (J, I) : NEXT
50 A(I) = POINT (I, I)
60 NEXT
70 FOR K = 1 TO 1000: NEXT : MODE (0)
80 FOR I = 1 TO 4
90 PRINT A(I)
100 NEXT
```

This example will draw lines on the screen colour with four colours. This command **POINT** will check the colour of that line and hold this information in the array A(I). Afterwards, the program will print out the colour of that line checked by the command **POINT**.

CHAPTER 17

COLOUR

- COLOR

COLOR

Your computer, as you already know, is capable of producing different colours. The instruction to set the colour is **COLOR I, J**, where I is the foreground colour – from 1 to 8 and J is the background colour – from Ø to 1.

In MODE (Ø)

Code	Colour
1	Green
2	Yellow
3	Blue
4	Red
5	Buff
6	Cyan
7	Magenta
8	Orange

The background colour can be either green (Ø) or orange (1). To change the background colour to orange, just type COLOR, 1. To get back to the original green just repeat using Ø instead of 1.

In MODE (1)

With background colour green (Ø), the foreground colours are:

Colour Code	Colour
1	green
2	yellow
3	blue
4	red

and with background colour buff (1), they become:

Colour Code	Colour
1	buff
2	cyan
3	magenta
4	orange

When the command COLOR I, J is executed, the foreground colour (graphics characters in TEXT mode and the points in GRAPHICS mode) is set to colour I and the background colour is set to colour J.

COLOR I will change the foreground colour to colour I with the background colour unchanged.

COLOR, J will change the background colour to colour J with the foreground colour unchanged.

Example:

```
10  COLOR 2, 0
```

It will set the foreground colour as yellow and the background colour as green.

Example:

```
10  COLOR 3, 1
```

It will set the foreground colour as blue and the background colour as orange.

Example:

```
10  COLOR 4
```

It will set the foreground colour as red with the background colour unchanged.

Example:

```
10  COLOR, 0
```

It will set the background colour as green with the foreground colour unchanged.

Example:

```
10 FOR I = 0 TO 15
20 FOR J = 1 TO 8
30 COLOR J
40 FOR K = 0 TO 3
50 PRINT TAB ((J–I)*4+K); "■";
60 NEXT : NEXT : NEXT
70 END
```

Note: By pressing SHIFT and J keys together you get the character "■".

This will display the eight colours that your computer is capable of production in this mode.

The colours in your computer are very suitable for graphics and game purpose especially in **MODE** (1).

CHAPTER 18

SOUND AND MUSIC

- SOUND
- MUSIC

SOUND

Another interesting feature of the computer is its ability to produce sound. Here is an example program.

Example:

```
1Ø FOR I = 1 TO 8
2Ø READ X
3Ø SOUND X, 7
4Ø NEXT I
5Ø DATA 16, 18, 2Ø, 21, 23, 25, 27, 28
   RUN
```

This will produce 8 notes going up the scale. In this program the variable X is the frequency and the constant 7 is the duration of the note.

It is possible to get 31 different frequencies and 9 different note durations. The tables below show how which codes produce the different frequencies and duration.

By varying the notes and duration therefore, and using the tables below, it is possible to produce tunes of your choice.

DURATION

Code	Note	Note length
1	𝅘𝅥𝅰	$\frac{1}{8}$
2	𝅘𝅥𝅯	$\frac{1}{4}$
3	𝅘𝅥𝅯.	$\frac{3}{8}$
4	𝅘𝅥𝅮	$\frac{1}{2}$
5	𝅘𝅥𝅮.	$\frac{3}{4}$
6	♩	1
7	♩.	$1\frac{1}{2}$
8	𝅗𝅥	2
9	𝅗𝅥.	3

FREQUENCY

Code	Pitch	Code	Pitch
0	rest	16	C4
1	A2	17	C#4
2	A#2	18	D4
3	B2	19	D#4
4	C3	20	E4
5	C#3	21	F4
6	D3	22	F#4
7	D#3	23	G4
8	E3	24	G#4
9	F	25	A4
10	F#3	26	A#4
11	G3	27	B4
12	G#3	28	C5
13	A3	29	C#5
14	A#3	30	D5
15	B3	31	D#5

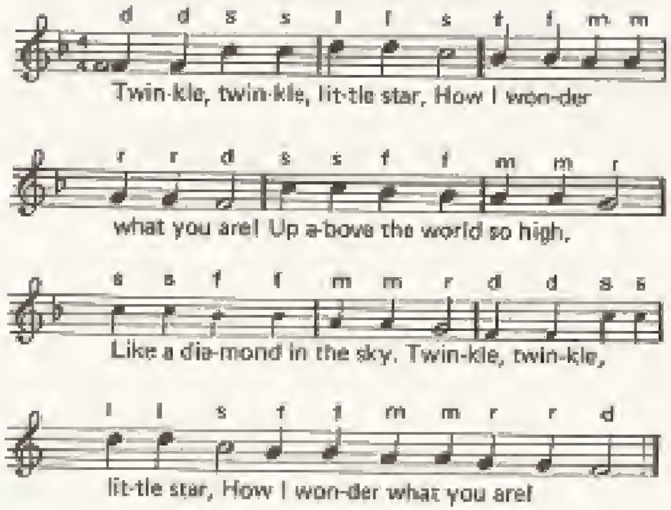
MUSIC

Below you can see how a musical score is transposed into Data for the computer.

TWINKLE, TWINKLE, LITTLE STAR

Nursery Rhyme

Key F



TWINKLE, TWINKLE, LITTLE STAR

Example:

```
2 DATA 21, 4, 21, 4, 28, 4, 28, 4, 30, 4, 30, 4, 28, 6, 26,
         4, 26, 4, 25, 4
4 DATA 25, 4 23, 4, 23, 4, 21, 6, 28, 4, 28, 4, 26, 4, 26,
         4, 25, 4, 25, 4, 23, 6
6 DATA 28, 4, 28, 4, 26, 4, 26, 4, 25, 4, 25, 4, 23, 6,
         21, 4, 21, 4, 28, 4 28, 4
8 DATA 30, 4, 30, 4, 28, 6, 26, 4, 26, 4, 25, 4, 25, 4,
         23, 4, 23, 4, 21, 6
10 FOR I = 1 TO 42: READ F, D: SOUND F, D:
NEXT: END
```

CHAPTER 19

MORE COMMANDS AND INFORMATION

- PRINT @
- PRINT TAB
- PRINT USING
- INP
- OUT
- USR

PRINT @

This command causes the **PRINT** output to be produced at a particular point on the screen. When dealing wih the **PRINT** @ command consider your screen to be a grid divided up into 32 X 16 spaces. Therefore there are 512 possible positions. The command takes this form

Example:

```
PRINT @ position, item list.
```

The position can be a number, variable or arithmetic expression. The value must be between Ø and 511.

Example:

```
6Ø PRINT @ 5Ø, 6ØØ;
```

Notice the use of the semi-colon at the end of the statement. This stops the rest of the line form being rubbed out.

PRINT TAB (expression)

This command works in very much the same way as the TAB on a typewriter. In this case it moves the cursor to a particular point on a line. The value of the expression must be between Ø to 255 inclusive. But note that for values of over 63, the cursor goes back to the start of the line and 'start again'. The same occurs at multiples of 64, so in practice only TAB values up to 63 should be used.

Example:

```
4ØPRINT TAB (6); 1; TAB (2Ø); 1
 RUN
      1                 1
```

PRINT USING string; item list

This command is useful in allowing you to state how you want your lines printed. It is very useful for reports and tables.

It takes the form of

Example:

```
PRINT USING string; value or string
```

This string or value can be either a variable or constant. What will happen is that the string or value to the right of the semi-colon will be inserted as specified by the field specifiers, the preceding string.

A) "!" This specifies that only the first character in the given string is to be printed.

Example:

```
10A$ = "ASDF"
20PRINT USING "!"; A$
RUN
A
```

B) " #" A number sign is used to represent each digit position in a numeric field. Digit positions are always filled. If the number to be printed has fewer digits than the positions specified, the number will be right justified (preceded by spaces) in the field.
"." A decimal point may be inserted at any position in the field. If the format string specifies that a digit is to precede the decimal point, the digit will always be printed (as Ø if necessary). Numbers are rounded as necessary.

Example:

```
PRINT USING "# #. # #"; .78
Ø.78
```

C) "+" A plus sign at the beginning or end of the format string will cause the sign of the number (plus or minus) to be printed before or after the number.

"–" A minus sign at the end of the format field will cause negative numbers to be printed with a trailing minus sign.

Examples:

```
PRINT USING "+ # #. # #"; –68.95
–68.95
PRINT USING "# # . # #–"; –68.95
68.95–
```

D) "**" A double asterisk at the beginning of the format string causes leading spaces in the numeric field to be filled with asterisks. The ** also specifies positions for two more digits.

Example:

```
PRINT USING "**#.#"; –Ø.9
*–Ø.9
```

E) "$$" A double dollar sign causes a dollar sign to be printed to the immediate left of the formatted number. The $$ specifies two more digit positions, one of which is the dollar sign. The exponential format cannot be used with $$. Negative numbers cannot be used unless the minus sign trails to the right.

Example:

```
PRINT USING "$$###.##"; 456.78
$456.78
```


F) "**$" The **$ at the beginning of a format string combines the effects of the above two symbols. Leading spaces will be asterisk-filled and a dollar sign will be printed before the number. **$ specifies three more digit positions, one of which is the dollar sign.

G) "," A comma that is to the left of the decimal point in a formatting string causes a comma to be printed to the left of every third digit to the left of the decimal point. A comma that is at the end of the format string is printed as part of the string. A comma specifies another digit position.

Example:

```
PRINT USING "####,.##"; 1234.5
1,234.5Ø
```

H) "%" If the number to be printed is larger than the specified numeric field, a percent sign is printed in front of the number. If rounding causes the number to exceed the field, a percent sign will be printed in front of the rounded number.

Examples:

```
PRINT USING "##.##"; 111.22
%111.22
PRINT USING ".##"; .999
%1.ØØ
```

INP (I)

This returns the byte read from input/output port I. I must be in the range Ø to 255. INP is the complementary function to the OUT statement (see below).

Example:

```
100 A = INP (255)
```

OUT I, J

This sends a byte to a machine output port. I and J are integer expressions in the range Ø to 255. I is the port number and J is the data to be transmitted.

Example:

```
100 OUT 32, 100
```


USR (X)

This calls the user's assembly language subroutine with the argument X. This subroutine could be taken from tape or made by POKEing instructions into memory locations. It is advised that you take great care when using this function as it can have disastrous effects on any stored program.

Example:

```
110 A = USR (8/2)
```

When this command is executed, the computer will get the content of the particular memory locations (Hexadecimal 788E–788F or decimal 3Ø 862–3Ø 863). These values will be taken as the starting address of the user's defined assembly language subroutine. That subroutine should be POKEd into the memory locations before the execution of the USR command. The value of the argument will also be passed to that subroutine. This value will be placed in the memory locations hexadecimal 7921–7922 (or decimal 31ØØ9–31Ø1Ø).

CHAPTER 20

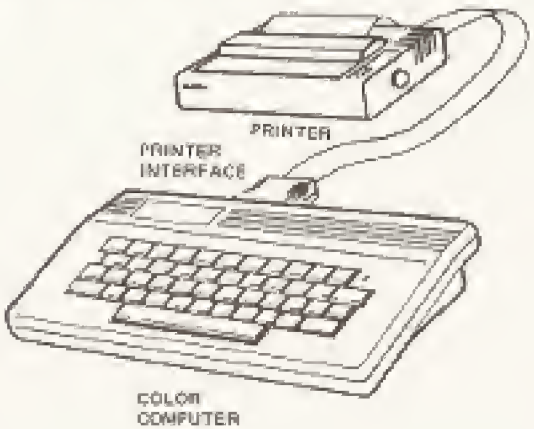
THE PRINTER

- LLIST
- LPRINT
- COPY

THE PRINTER (OPTIONAL)

To further expand the capabilities of your computer you can acquire a PRINTER. This can be attached to your computer by means of a PRINTER INTERFACE. If you decide to acquire a PRINTER, you will receive a separate leaflet containing detailed operating instructions.

SETTING UP THE PRINTER



To operate the computer successfully you need to familiar with the following commands LLIST, LPRINT, COPY.

LLIST

This performs a similar function in relation to the PRINTER as LIST does in relation to the TV screen. LLIST outputs to PRINTER, to give you a "hard copy" of your programs.

LPRINT

This command (and statements) is similar to PRINT, except that LPRINT is used with the PRINTER.

COPY

This command will print out what you see on the video screen to the PRINTER. You can stop the PRINTER by pressing the BREAK [CTRL- –].

NOTE: If the screen is dislplaying graphics in either Text mode or Graphics mode, you can only use the COPY command if you are using a DSE GP-100 CAT No. X-3250 printer. This also happens for the LPRINT and LLIST commands to send graphics characters to the printer.
Other Centronics-type printers can be used but only if you do not try to print out graphics characters.

CHAPTER 21

HINTS ON PROGRAMMING

- EFFECTIVE PROGRAMMING
- DOUBLE PRECISION ARITHMETICS

EFFECTIVE PROGRAMMING

There are several methods that you can use to improve your program. A program is effective means that this program occupies less memory space or executes much faster. The methods are explained as follows:

1) Use fewer LET

The use of the command **LET** is optional for this computer. You may omit the **LET** command to save memory space for your program.

2) Use fewer REM

REM command will allow you add remark to your program but it also occupies memory space. You may delete all unnecessary **REM** statements in your program.

3) **Use fewer space**

You may delete the unnecessary space between statements, operators, etc.

Example:

```
10 FOR I = 1 TO 10
```

is the same as

Example:

```
10FORI=1TO10
```

but saves some memory space.

4) **Use Multiple-Statements**

The use of multiple-statements will save memory space and also speed up your program.

5) **Use integer variables**

Integer variables occupy less space than real variables. Whenever possible, you may substitute the real variable A, for example, by the integer variable A%.

6) **Use subroutines**

You may use subroutines to save program space if the operation is called from different places on your program several times.

7) **Use fewer parentheses**

You may eliminate the use of parentheses whenever possible to save memory space.

8) Use simple expression in functions

You may get the result from a function much faster if you simplify the argument of that function.

Example:
```
10 A = 3↑2 + 17 – 22/3 : A = INT (A)
```

will execute much faster than

Example:
```
10 A = INT (3↑2 + 17 – 22/3)
```

but occupy some more space.

9) Estimate the size of Array

The use of **DIM** occupies many memory space. Therefore it is better to assign just enough size for your array. Also use the zero subscripted elements as they are always available.

10) Use **READ** and **DATA**

Often you have to use the same command many times but with different parameters. In order to save space and repetitive typing, you may put these parameters into **DATA** statements and retrieve them by **READ** statements.

11) Use **CLEAR**

You may find there is not enough room for string variables. Use the **CLEAR** command with a larger value to assign more space for strings.

DOUBLE PRECISION ARITHMETICS

This is a recommended method for those users who want to use the computer to do double precision arithmetics. The simple example below shows the implementation.

Example:
```
10 CLEAR 100
20 A$ = STR$ (10/9 #)
30 PRINT "A=";A$
40 B$=STR$(VAL(A$)*VAL(A$))
50 PRINT "A*A=";B$
```

In line 10, the program reserves 100 bytes for string operation. In line 20, the variable is defined as a string with a equivalent numeric value of 10/9. The '#' sign following 10/9 tells the computer that the value of 10/9 should be in double precision representation.

Line 30 prints out the value of 10/9 in double precision value. In line 40 and 50, the value of 10/9 * 10/9 is calcuculated and printed out in double precision value.

Type "RUN" and the computer will have the result as following.

```
RUN
A=1.111111111111111
A*A=1.234567901234568
```

APPENDIX

- ERROR MESSAGES
- ASCII CODE TABLE
- CHARACTER CODE
- SUMMARY OF BASIC COMMANDS
- QUICK REFERENCE OF BASIC COMMANDS

Number ERROR MESSAGES

1 BAD FILE DATA (BAD FILE DATA)
The type of data stored on tape by PRINT # does not match with the type of variables read by INPUT #.

2 CANNOT CONTINUE (CAN'T CONT)
An attempt is made to continue a program that:
1. has halted due to an error,
2. has been modified during a break in execution, or
3. does not exist.

3 DISK COMMAND (DISK COMMAND)
An attempt to use DISK COMMAND without the connection of DISK DRIVE.

4 DIVISION BY ZERO (DIVISION BY ZERO)
A division by zero is encountered in an expression, or the operation of involution results in zero being raised to a negative power.

5 ILLEGAL DIRECT (ILLEGAL DIRECT)
A statement that is illegal in direct mode is entered as a direct mode command. Example: INPUT

6 ILLEGAL FUNCTION CALL (FUNCTION CODE)
A parameter that is out of range is passed to a math or string function. This error may also occur as the result of:
1. a negative or unreasonably large subscript
2. a negative or zero argument with LOG
3. a negative argument of SQR
4. a negative mantissa with a non-integer exponent
5. a call to a USR function for which the starting address has not yet been given
6. an improper argument to MID$, LEFT$, RIGHT$, INP, OUT, PEEK, POKE or TAB.

7 LOADING ERROR (LOADING ERROR)
The program loaded into the computer is incorrect. (Refer to chapter STORING YOUR PROGRAMS ON TAPE)

8 MISSING OPERAND (MISSING OPERAND)
The operand of some commands are missed. Example: COLOR, MODE.

9 NEXT WITHOUT FOR (NEXT WITHOUT FOR)
A variable in a NEXT statement does not correspond to any previously executed, unmatched FOR statement variable.

10 OUT OF DATA (OUT OF DATA)
A READ statement is executed when there are no DATA statements with unread data remaining in the program.

11 OUT OF MEMORY (OUT OF MEMORY)
A program is too large, has too many FOR loops or GOSUBs, too many variables, or expressions that are too complicated.

12 OUT OF STRING SPACE (OUT OF SPACE)
String variables have caused BASIC to exceed the amount of free memory remaining.

13 OVERFLOW (OVERFLOW)
The result of a calculation is too large to be represented in the number format. If underflow occurs, the result is zero and execution continues without an error.

14 REDIMENSIONED ARRAY (REDIM'D ARRAY)
Two DIM statements are given for the same array, or a DIM statement is given for an array after the default dimension of 10 has been established for that array.

15 REDO (REDO)
A string is assigned to a numeric variable during the execution of the INPUT command.

16 RETURN WITHOUT GOSUB (RET'N WITHOUT GOSUB)
A RETURN statement is encountered for which there is no previous, unmatched GOSUB statement.

17 STRING FORMULA TOO COMPLEX (FORMULA TOO COMPLEX)
A string expression is too long or too complex. The expression should be broken into smaller expressions.

18 STRING TOO LONG (STRING TOO LONG)
An attempt is made to create a string more than 255 characters long.

19 SUBSCRIPT OUT OF RANGE (BAD SUBSCRIPT)
An array element is referenced either with a subscript that is outside the dimensions of the array, or with the wrong number of subscripts.

20 SYNTAX ERROR (SYNTAX)
A line is encountered that contains some incorrect sequence of characters (such as unmatched parenthesis, misspelled command or statement, incorrect punctuation, etc.).

21 TYPE MISMATCH (TYPE MISMATCH)
A string variable name is assigned a numeric value or vice versa; a function that expects a numeric argument is given a string argument or vice versa.

22 UNDEFINED STATEMENT (UNDEF'D STATEMENT)
A line reference in a GOTO, GOSUB or IF . . . THEN . . . ELSE is to a nonexistent line.

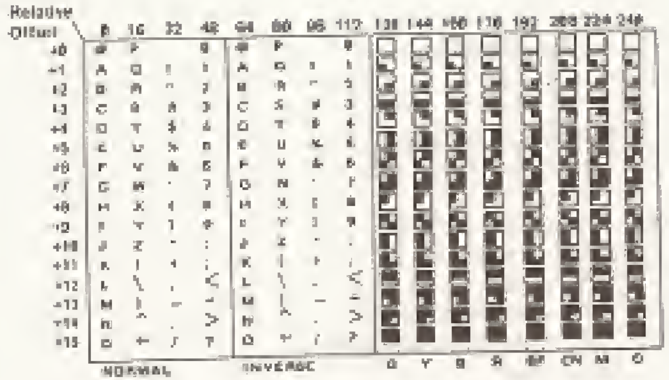
23 VERIFY ERROR (VERIFY ERROR)
The program on tape is not indentical to the program in memory. (Refer to chapter STORING YOUR PROGRAMS ON TAPE)

Note: All enclosed statements are those messages that will appear on the screen.

ASCII CODE TABLE

ASCII CODE	CHARACTER	ASCII	CHARACTER
32	(Space)	64	@ (at sign)
33	! (exclamation point)	65	A
34	" (quote)	66	B
35	# (number or pound sign)	67	C
36	$ (dollar)	68	D
37	% (percent)	69	E
38	& (ampersand)	70	F
39	' (apostrophe)	71	G
40	((open parenthesis)	72	H
41	) (close parenthesis)	73	I
42	* (asterisk)	74	J
43	+ (plus)	75	K
44	, (comma)	76	L
45	– (minus)	77	M
46	. (period)	78	N
47	/ (slant)	79	O
48	0	80	P
49	1	81	Q
50	2	82	R
51	3	83	S
52	4	84	T
53	5	85	U
54	6	86	V
55	7	87	W
56	8	88	X
57	9	89	Y
58	: (colon)	90	Z
59	; (semicolon)	91	[(open square bracket)
60	< (less than)	92	\ (back slash)
61	= (equals)	93	] (close square bracket)
62	> (greater than)	94	↑ (up arrow)
63	? (question mark)	95	← (back arrow)

CHARACTER CODE



G – GREEN
Y – YELLOW
B – BLUE
R – RED

BF – BUFF
CN – CYAN
M – MAGENTA
O – ORANGE

SUMMARY OF BASIC COMMANDS

Functions:

1) Arithmetic operators
   +, –, *, /, ↑

2) Relational operators
   >, <, =, >=, <=, <>

3) Arithmetic functions:
   SQR – Square root
   INT – Integer part
   RND – Random number
   ABS – Absolute magnitude
   SGN – Sign
   COS – Cosine
   SIN – Sine
   TAN – Tangent
   ATN – Arc tangent
   EXP – e^x
   LOG – Natural logarithm

4) String functions:
   LEN – Length
   STR$ – String of numeric argument
   VAL – Numeric value of string
   ASC – ASCII value
   CHR$ – Character
   LEFT$ – Left characters
   MID$ – Middle characters
   RIGHT$ – Right characters
   INKEY$ – Check keyboard

5) Logical operators
   AND
   OR
   NOT
   Relational and logical expressions have value 1 if true, Ø if false.

6) Graphics and sound functions:
   CLS – Clear screen
   SET – Plot a point
   RESET – Clear a point
   POINT – Return the color code if graphics point set.
   COLOR – Set colour
   SOUND – Produce tone of specified frequency and duration
   MODE – Select graphics (1) or text mode (Ø)

7) Program statements
   DIM – Dimension array
   STOP
   END
   GOTO
   GOSUB
   RETURN
   FOR . . . TO . . . STEP
   NEXT
   REM
   IF . . . THEN . . . ELSE
   INPUT
   INPUT #
   PRINT
   PRINT #
   PRINT TAB
   PRINT USING

PRINT @
LET
DATA
READ
RESTORE

8) Commands:
   LIST
   RUN
   NEW
   CONT
   VERIFY — Check whether program on tape and memory are identical
   CLOAD — Load program from tape
   CSAVE — Save program on tape
   CRUN — Load program from tape and run
   $\boxed{\text{BREAK}}$ (CTRL- —) — To halt program

9) Other Statements
   PEEK — Return the value stored at the location specified
   POKE — Load a value into a specified location
   LPRINT — Print on printer
   LLIST — List on printer
   INP — Return the content read from port
   OUT — Send values to an I/0 port
   COPY — Copy the content of screen to printer
   USR — Call the user's assembly language subroutine

QUICK REFERENCE OF BASIC COMMANDS

NOTE: arg — argument (constant or variable)
var — variable
cont — constant
str — string
str arg — string constant or variable
lin — line number
exp — expression

NO.	COMMANDS	MEANING & SYNTAX	PAGE
1	ABS (arg)	— Absolute magnitude	78
2	AND	— Relational and logical expression, have value 1 if true, Ø if false.	102
3	ASC (str arg)	— ASCII value in decimal	89
4	ATN (arg)	— Arctangent function — Result in radians — −1ØE38 ≤ arg ≤ 1ØE38	78
5	CHR$ (arg)	— Character represented by ASCII code — Ø ≤ arg ≤ 255	90
6	CLEAR arg	— Clear space for variable	74
7	CLOAD "file name"	— Load program from tape — Only first 16 characters of "file name" are vaild	145
8	CLS	— Clear screen	37

9	COLOR arg 1, arg 2	– Set colour – arg 2 = Ø or 1 – In graphic mode 1 ≤ arg 1 ≤ 4 In text mode 1 ≤ arg 1 ≤ 8	165
10	CONT	– Continue program execution	71
11	COPY	– Copy the content on screen to printer	190
12	COS (arg)	– Cosine function – arg in radians – –9999999 ≤ arg ≤ 9999999	78
13	CRUN "file name"	– Load program from tape and run – Only first 16 characters of "file name" are vaild	151
14	CSAVE "file name"	– Save program on tape – Only first 16 characters of "file name" are vaild	147
15	DATA cont 1 [str 1] , . . .	– Assign value for READ command	129
16	DIM var (arg)	– Assign size of array – Another form: DIM var$ (arg) – Up to 3 dimensional array	123
17	ELSE	– Refer IF	97
18	END	– Terminate program	73
19	EXP (arg)	– Exponential function – e = 2.71828 – – 1Ø E 38 ≤ arg ≤ 87	78

2Ø	FOR var = arg 1 TO arg 2 STEP arg 3	– Perform looping – arg 1 ≤ arg 2 – arg 3 ≠ Ø	1Ø8
21	GOSUB lin	– Go to subroutine	117
22	GOTO lin	– Jump to specified line number	69
23	IF exp 1 THEN exp 2 ELSE exp 3	– Conditional branching – exp 1, exp 2 and exp 3 may be logical expressions – exp 2 and exp 3 may be line numbers	97
24	INKEY$	– Check keyboard – General form: var$ = INKEY$	93
25	INP (arg)	– Return content read from Port arg – Ø ≤ arg ≤ 255 – General form: var = INP (arg)	185
26	INPUT str var 1 [var 1 $] , ...	– Print out str and ask assigned to var [var$]	6Ø
27	INPUT # "file name" var 1 [var 1$] , ...	– Read the values of var 1 (var1$) etc. from the date file "file name".	152
28	INT (arg)	– Get integer part – –1Ø E 38 ≤ arg ≤ 1Ø E38	78
29	LEFT$ (str arg, arg)	– Left arg charcters	88

3Ø	LEN (str arg)	– Length of string	85
31	LET var = numeric exp	– Assign value – Other form: LET var$= str exp	51
32	LIST lin 1 – lin 2	– List from lin 1 to lin 2 – lin 2 ≥ lin 1 – Other form : LIST : LIST lin 1 – : LIST – lin 2	64
33	LLIST lin 1 – lin 2	– Similar to LIST – List to printer	190
34	LOG(arg)	– Natural Logarithm – arg > Ø	78
35	LPRINT arg 1 [str arg 1] , . . .	– Similar to PRINT – Print to printer – ',' or ';' may be used	190
36	MID$ (str arg, . arg 1, arg 2)	– Middle arg 2 characters starting from arg 1TH character.	89
37	MODE (arg)	– Select graphics or text – arg = Ø or 1	157
38	NEW	– Clear screen and memory	61
39	NEXT	– Refer FOR	109
4Ø	NOT	– Relational and logical expression have value 1 if true, Ø if false	102
41	OR	– Relational and logical expression have value 1 if true, Ø if fasle	102
42	OUT (arg 1, arg)	– Sent arg 2 to Port arg 1 – Ø ≤ arg 1, arg 2 ≤ 255	185

43	PEEK (arg)	– Return decimal value stored at memory loca- tion arg – –32768 ≤ arg ≤ 32767	135
44	POINT (arg 1, arg 2)	– Return colour code – Ø ≤ arg 1 ≤ 127 Ø ≤ arg 2 ≤ 63	160
45	POKE arg 1, arg 2	– Load arg 2 into memory location arg 1 – Ø ≤ arg 2 ≤ 255 –32768 ≤ arg 1 ≤ 32767	136
46	PRINT arg 1 [str arg 1] , . .	– Print arg or str arg on screen – Use "," or ";"	28
47	PRINT TAB (arg 1) arg 2 [str arg 2]	– Similar to PRINT – Tabulate arg 1 space – Ø ≤ arg 1 ≤ 255	180 180
48	PRINT USING str; arg 1, . . .	– Similar to PRINT – Output in specified for- mat	
49	PRINT @ arg 1, arg 2 [str arg 2] , . . .	– Similar to PRINT – Start printing arg 2 etc at location arg 1 – Ø ≤ arg 1 ≤ 511	179
5Ø	PRINT # "file name" var 1 [var 1 $] , . . .	– Send out the contents of var 1 (var 1 $) etc. . . to the date file "file name".	151
51	READ var 1 [var 1 $] , . . .	– Assign value in DATA statement to variable	129
52	REM exp	– Remark	60

53	RESET (arg 1, arg 2)	– Clear a point – $\emptyset \leqslant$ arg 1 $\leqslant$ 127 $\emptyset \leqslant$ arg 2 $\leqslant$ 63	160
54	RESTORE	– Recover the same DATA	130
55	RETURN	– Terminate subroutine	117
56	RIGHT$ (str arg 1, arg 2)	– Right arg 2 characters	
57	RND (arg)	– Generate random number – arg $\geqslant \emptyset$	78
58	RUN lin	– Start execution at lin – Other form: RUN	63
59	SET (arg 1, arg 2)	– Plot a point – $\emptyset \leqslant$ arg 1 $\leqslant$ 127 $\emptyset \leqslant$ arg 2 $\leqslant$ 63	159
60	SGN (arg)	– Sign of arg	78
61	SIN (arg)	– Sine function – arg in radians – $-9999999 \leqslant$ arg $\leqslant$ 9999999	78
62	SOUND arg 1, arg 2	– Sound – $\emptyset \leqslant$ arg 1 $\leqslant$ 31 1 $\leqslant$ arg 2 $\leqslant$ 9	171
63	SQR (arg)	– Square root – arg $\geqslant \emptyset$	78
64	STEP	– Refer FOR	109
65	STOP	– Halt program	72
66	STR$ (arg)	– Change arg to string	86

67	TAN (arg)	– Targent function – arg in radians – $-9999999 \leqslant$ arg $\leqslant$ 9999999	78
68	THEN	– Refer IF	97
69	TO	– Refer FOR	109
70	USR (arg)	– CAll user's assembly lanugage subroutine – General form: var = USR (arg)	186
71	VAL (str arg)	– Numeric value of string – str arg should be numeric string	87
72	VERIFY "file name"	– Compare program on tape and program on memory – Only first 16 characters of "file name" are valid	149

PART III
BASIC APPLICATION PROGRAMS

PART III BASIC APPLICATION PROGRAMS

TABLE OF CONTENT

1. SUM & AVERAGE

This program computes the total sum and average of a group of numbers. Can you tell the logic behind the computer?

```
10 REM SUM & AVERAGE
20 CLS
30 PRINT "SUM AND AVERAGE"
40 INPUT "ENTER HOW MANY NOS.";A
50 FOR I = 1 TO A
60 PRINT "NOS.";I;"=";
70 INPUT B
80 C=C+B : NEXT
90 PRINT "SUM =";C
100 PRINT "AVERAGE =";C/A
110 END

RUN

SUM AND AVERAGE
ENTER HOW MANY NOS.? 5
NOS. 1 =? 10
NOS. 2 =? 20
NOS. 3 =? 30
NOS. 4 =? 40
NOS. 5 =? 50
SUM = 150
AVERAGE = 30
READY
```


2. PERMUTATION & COMBINATION

Permutation and combination are 2 popular subjects in modern mathematics. By using this program, you can get the answers quickly. Can you beat the computer in speed and accuracy?

```
10 REM PERMUTATION &
20 REM COMBINATION
30 CLS
40 PRINT "PERMUTATION & ";
50 PRINT "COMBINATION"
60 INPUT "ENTER TOTAL NOS.";A
70 INPUT "ENTER SUBSET NOS.";B
80 C=1 : D=1
90 IF B>A THEN 30
100 FOR I = A-B+1 TO A
110 IF C*I>1E36 THEN 200
120 C=C*I : NEXT
130 FOR I = 2 TO B
140 D=D*I : NEXT
150 PRINT "PERMUTATION =";C
160 PRINT "COMBINATION =";C/D
170 END
200 PRINT "OVERFLOW" : GOTO60

RUN

PERMUTATION & COMBINATION
ENTER TOTAL NOS.? 5
ENTER SUBSET NOS.? 4
PERMUTATION = 120
COMBINATION = 5
READY
```

3. HIGHEST COMMON FACTOR (H.C.F.)

Just input 2 numbers and this program will tell you the Highest Common Factor.

```
10 REM FIND HCF
20 CLS
30 PRINT "FIND H.C.F."
40 INPUT "ENTER 2 NUMBERS";A,B
50 IF A=0 OR B=0 THEN 100
60 IF A>B THEN A=A-B
70 IF A<B THEN B=B-A
80 IF A<>B THEN 60
90 PRINT "H.C.F. =";A
100 END

RUN

FIND H.C.F.
ENTER 2 NUMBERS? 20
?? 10
H.C.F. = 10
READY
```


4. LOWEST COMMON MULTIPLE (L.C.M.)

Similar to (H.C.F.) but will give you the Lowest Common Multiple instead of the Highest Common Factor.

```
10 REM FIND LCM
20 CLS
30 PRINT "FIND L.C.M."
40 INPUT "ENTER 2 NUMBERS";A,B
50 IF A=0 OR B=0 THEN 110
60 IF A>B THEN C=A-1 ELSE C=B-1
70 C=C+1
80 IF INT(C/A)<>C/A THEN 70
90 IF INT(C/B)<>C/B THEN 70
100 PRINT "L.C.M. =";C
110 END

RUN

FIND L.C.M.
ENTER 2 NUMBERS? 11
?? 13
L.C.M. = 143
READY
```

5. PRIME FACTOR

This program identifies all the prime factors hidden in any number.

```
10 REM PRIME FACTORS
20 CLS
30 PRINT "PRIME FACTORS"
40 INPUT "ENTER A NUMBER";A
50 IF A=0 THEN 130
60 PRINT SGN(A); : A=ABS(A)
70 FOR I = 2 TO A : B=0
80 IF A/I<>INT(A/I) THEN 100
90 A=A/I : B=B+1 : GOTO 80
100 IF B=0 THEN 120
110 PRINT I;"^";B;
120 NEXT
130 END

RUN

PRIME FACTORS
ENTER A NUMBER? 240
 1  2 ^ 4  3 ^ 1  5 ^ 1
READY
```


6. ROOTS OF QUADRATIC EQUATION

Generally speaking, Quadratic Equations are in the form of ax^2+bx+c=0, where a, b, and c, are the constant coefficients and x is the unknown variable. This program can find out the roots (values of x) for you easily.

```
10 REM ROOTS OF QUADRATIC
20 REM EQUATION
30 CLS
40 PRINT "QUADRATIC EQUATION"
50 PRINT "A*X^2+B*X+C=0"
60 PRINT "ENTER COEFFICIENTS ";
70 PRINT "A,B,C"
80 INPUT A,B,C
90 D=B^2-4*A*C
100 IF D<0 THEN 160
110 D=SQR(D)
120 PRINT "THE ROOTS ARE :"
130 PRINT (-B-D)/(2*A);
140 PRINT (-B+D)/(2*A)
150 GOTO 170
160 PRINT "NO REAL ROOTS"
170 END

RUN

QUADRATIC EQUATION
A*X^2+B*X+C=0
ENTER COEFFICIENTS A,B,C
? 1
?? 1
?? -12
THE ROOTS ARE :
-4  3
READY
```

7. AREA OF TRIANGLE

The area of a triangle can be determined once the three sides are fixed. Can you write a program to find out the area of a circle if I can give you the radius.

```
10 REM AREA OF TRIANGLE
20 CLS
30 PRINT "AREA OF TRIANGLE"
40 PRINT "ENTER 3 SIDES"
50 INPUT A,B,C
60 D=.5*(A+B+C)
70 E=D*(D-A)*(D-B)*(D-C)
80 PRINT "AREA IS";SQR(E)
90 END

RUN

AREA OF TRIANGLE
ENTER 3 SIDES
? 6
?? 8
?? 10
AREA IS 24
READY
```


8. AREA OF POLYGON

In this program, the area of a regular polygon can be computed. All you have to do is to input the number of sides and its corresponding length.

```
10 REM AREA OF POLYGON
20 CLS : PI=3.1416
30 PRINT "AREA OF REGULAR ";
40 PRINT "POLYGON"
50 INPUT "ENTER NOS. OF SIDES";A
60 INPUT "ENTER LENGTH";B
70 C=PI*(.5*A-1)/A
80 D=A*B*B*TAN(C)/4
90 PRINT "AREA IS";D
100 END

RUN

AREA OF REGULAR POLYGON
ENTER NOS. OF SIDES? 5
ENTER LENGTH? 4
AREA IS 27.5278
READY
```

9. RADIAN & DEGREE

This program converts any values in radians to degrees, and vice versa.

```
10 REM RADIAN & DEGREE
20 CLS
30 INPUT "FIND RADIAN(1) OR DEGREE(2)";S
40 IF S=1 THEN 140
50 INPUT "RADIAN";B
60 C=B*180/3.1416
70 IF C>360 THEN C=C-360 : GOTO 70
80 PRINT INT(C);"DEGREES"
90 D=(C-INT(C))*60
100 PRINT INT(D);"MINUTES"
110 E=(D-INT(D))*60
120 PRINT INT(E);"SECONDS"
130 END
140 INPUT "DEGREES";A
150 INPUT "MINUTES";B
160 INPUT "SECONDS";C
170 PRINT
180 D=A+B/60+C/3600
190 IF D>360 THEN D=D-360 : GOTO190
200 D=D*3.1416/180
210 PRINT D;"RADIANS"
220 END

RUN

FIND RADIAN(1) OR DEGREE(2)? 1
DEGREES? 1
MINUTES? 1
SECONDS? 1

 .0177491 RADIANS
READY
```



```
RUN

FIND RADIAN(1) OR DEGREE(2)? 2
RADIAN? 1
 57 DEGREES
 17 MINUTES
 44 SECONDS
READY
```

10. FAHRENHEIT & CELSIUS

Similar to Radian & Degree, except to tell you the conversion in temperature.

```
10 REM DEGREE FAHRENHEIT &
20 REM CELSIUS
30 CLS
40 PRINT "FIND DEGREE-F(1)"
50 INPUT "   OR DEGREE-C(2)";A
60 IF A=2 THEN 110
70 INPUT "DEGREE-C";B
80 PRINT B;"DEGREE-C =";
90 PRINT B*9/5+32;"DEGREE-F"
100 END
110 INPUT "DEGREE-F";B
120 PRINT  B;"DEGREE-F =";
130 PRINT (B-32)*5/9;"DEGREE-C"
140 END

RUN

FIND DEGREE-F(1)
  OR DEGREE-C(2)? 1
DEGREE-C? 0
 0 DEGREE-C = 32 DEGREE-F
READY


RUN

FIND DEGREE-F(1)
  OR DEGREE-C(2)? 2
DEGREE-F? 32
 32 DEGREE-F = 0 DEGREE-C
READY
```


11. FOOT & METRE

Similar to Radian & Degree, except the subjects are Foot & Metre.

```
10 REM FOOT & METRE
20 CLS
30 PRINT "FIND FOOT(1) OR ";
40 INPUT "METRE(2)";A
50 IF A=1 THEN 100
60 INPUT "FEET";B
70 PRINT B;"FEET =";
80 PRINT .3048*B;"METRES"
90 END
100 INPUT "METRES";B
110 PRINT B;"METRES =";
120 PRINT B/.3048;"FEET"
130 END

RUN

FIND FOOT(1) OR METRE(2)? 1
METRES? 1
 1 METRES = 3.28084 FEET
READY


RUN

FIND FOOT(1) OR METRE(2)? 2
FEET? 1
 1 FEET = .3048 METRES
READY
```

12. POUND & KILOGRAM

Similar to Radian & Degree, except that Pound & Kilogram are being converted.

```
10 REM POUND & KILOGRAM
20 CLS
30 INPUT "FIND POUND(1) OR KILOGRAM(2)";A
40 IF A=1 THEN 90
50 INPUT "POUNDS";B
60 PRINT B;"POUNDS =";
70 PRINT .4536*B;"KILOGRAMS"
80 END
90 INPUT "KILOGRAMS";B
100 PRINT B;"KILOGRAMS =";
110 PRINT B/.4536;"POUNDS"
120 END

RUN

FIND POUND(1) OR KILOGRAM(2)? 1
KILOGRAMS? 1
 1 KILOGRAMS = 2.20459 POUNDS
READY

RUN

FIND POUND(1) OR KILOGRAM(2)? 2
POUNDS? 1
 1 POUNDS = .4536 KILOGRAMS
READY
```


13. GALLON & LITRE

Similar to Radian & Degree, except that Gallon & Litre are used.

```
10 REM GALLON & LITRE
20 CLS
30 INPUT "FIND GALLON(1) OR LITRE(2)";A
40 IF A=1 THEN 90
50 INPUT "GALLONS";B
60 PRINT B;"GALLONS =";
70 PRINT 4.546*B;"LITRES"
80 END
90 INPUT "LITRES";B
100 PRINT B;"LITRES =";
110 PRINT B/4.546;"GALLONS"
120 END

RUN

FIND GALLON(1) OR LITRE(2)? 1
LITRES? 1
 1 LITRES = .219974 GALLONS
READY

RUN

FIND GALLON(1) OR LITRE(2)? 2
GALLONS? 1
 1 GALLONS = 4.546 LITRES
READY
```

14. DEPRECIATION

The value of most commodities will decrease after a certain period of time. This program calculates the depreciation value (the difference) once you have input the original price, the depreciation rate and the timing involved.

```
10 REM DEPRECIATION
20 CLS
30 INPUT "ORIGINAL PRICE";A
40 INPUT "DEPRECIATION RATE(%)";B
50 INPUT "NO. OF YEARS";C
60 PRINT "DEPRECIATION =";
70 B=B/100
80 D=A*B*(1-B)^(C-1)
90 D=INT(D*10+.5)/10
100 PRINT D : END

RUN

ORIGINAL PRICE? 1000
DEPRECIATION RATE(%)? 10
NO. OF YEARS? 5
DEPRECIATION = 65.6
READY
```


15. SORTING NUMBERS

If you input a group of numbers (from 2 to 20), this program will sort the numbers in an ascending order. Can you modify the program in the way that it can sort the numbers in a descending order?

```
10 REM SORTING NOS. IN
20 REM ASCENDING ORDER
30 CLS
40 PRINT "SORTING NOS.(2-20)"
50 INPUT "HOW MANY NOS.";A
60 DIM A(19)
70 FOR I = 1 TO A
80 PRINT "NO.";I; : INPUT A(I-1)
90 NEXT
100 FOR J = 0 TO A-2
110 FOR I = 0 TO A-2
120 IF A(I)<A(I+1) THEN 140
130 B=A(I) : A(I)=A(I+1) : A(I+1)=B
140 NEXT : NEXT
150 FOR I = 0 TO A-1
160 PRINT A(I);
170 NEXT
180 END

RUN

SORTING NOS.(2-20)
HOW MANY NOS.? 6
NO. 1 ? 6
NO. 2 ? 5
NO. 3 ? 4
NO. 4 ? 3
NO. 5 ? 2
NO. 6 ? 1
 1  2  3  4  5  6
READY
```

16. SORTING WORDS

This program sorts a group of words (from 2 to 10) in an alphabetic order.

```
10 REM SORTING WORDS IN
20 REM ALPHABETIC ORDER
30 CLS
40 PRINT "SORTING WORDS(2-10)"
50 INPUT "HOW MANY WORDS";A
60 DIM A$(9)
70 FOR I = 1 TO A
80 PRINT "WORD";I; : INPUT A$(I-1)
90 NEXT
100 FOR J = 0 TO A-2
110 FOR I = 0 TO A-2
120 IF A$(I)<A$(I+1) THEN 140
130 B$=A$(I) : A$(I)=A$(I+1) : A$(I+1)=B$
140 NEXT : NEXT
150 FOR I = 0 TO A-1
160 PRINT A$(I);" ";
170 NEXT
180 END
```

```
RUN

SORTING WORDS(2-10)
HOW MANY WORDS? 6
WORD 1 ? ZOO
WORD 2 ? FAST
WORD 3 ? LAZY
WORD 4 ? EAT
WORD 5 ? EAR
WORD 6 ? HELLO
EAR EAT FAST HELLO LAZY ZOO
READY
```


17. NUMBER GUESSING

The computer will generate a number in random (from 1 to 1000) and you will have to guess what is the pre-selected number. How many trials you have to attempt?

```
10 REM GUESS A NUMBER
20 CLS : C=1
30 A=RND(1000)
40 PRINT "GUESS A NUMBER"
50 INPUT "(1-1000)";B
60 IF B>A THEN PRINT "SMALLER"
70 IF B<A THEN PRINT "LARGER"
80 IF B=A THEN 100
90 C=C+1 : GOTO 40
100 PRINT "YOU ARE RIGHT"
110 PRINT "YOU HAVE TRIED";C;
120 PRINT "TIMES"
130 END
```

```
RUN

GUESS A NUMBER
(1-1000)? 500
SMALLER
GUESS A NUMBER
(1-1000)? 250
LARGER
GUESS A NUMBER
(1-1000)? 300
YOU ARE RIGHT
YOU HAVE TRIED 3 TIMES
READY
```

This is only an example. The numbers may vary as the program uses random numbers.

18. WORD GUESSING

This time you have to guess a 4 letter word. The method of playing is similar to the Number Guessing.

```
10 REM GUESS A WORD
20 CLS
30 C$="FISHRUSHRESTSIDETALKDIRTWORKGIRLJUMPMOOD"
40 I=(RND(10)-1)*4+1
50 A$=MID$(C$,I,4) : S=1
60 PRINT "GUESS A WORD"
70 INPUT "(4 LETTERS)";B$
80 FOR J = 1 TO 4
90 IF MID$(A$,1,J)=MID$(B$,1,J) THEN NEXT
100 PRINT "YOU HAVE";J-1;
110 PRINT "LETTERS RIGHT"
120 IF J<>5 THEN S=S+1 : GOTO 60
130 PRINT "YOU HAVE TRIED";S;
140 PRINT "TIMES"
150 END

RUN

GUESS A WORD
(4 LETTERS)? R
YOU HAVE 0 LETTERS RIGHT
GUESS A WORD
(4 LETTERS)? S
YOU HAVE 1 LETTERS RIGHT
(4 LETTERS)? SIDE
YOU HAVE 4 LETTERS RIGHT
YOU HAVE TRIED 3 TIMES
READY
```

This is only an example. The letters may vary as the program chooses them randomly.

19. RANDOM GRAPHICS

Random graphic is generated by making use of the pre-defined graphic characters.

```
10 REM GRAPHIC
20 CLS
30 COLOR RND(8)
40 PRINT@ RND(512)-1,"█";
50 GOTO 30
```

20. MELODY

You can write and play your own song. All you have to do is to select the frequency code and the duration code of each note. However, the maximum number of notes that you can play at one time will depend on the memory size of your computer.

```
10 REM SONG
20 CLS
30 INPUT "ENTER NO. OF NOTES";N
40 PRINT "ENTER YOUR NOTES"
50 DIM A%(2*N-1)
60 FOR I = 0 TO N-1
70 INPUT "FREQUENCY CODE";A%(I*2)
80 INPUT "DURATION CODE";A%(I*2+1)
90 NEXT
100 FOR I = 0 TO N-1
110 SOUND A%(I*2),A%(I*2+1)
120 NEXT

RUN

ENTER NO. OF NOTES? 8
ENTER YOUR NOTES
FREQUENCY CODE? 26
DURATION CODE? 3
FREQUENCY CODE? 30
DURATION CODE? 3
FREQUENCY CODE? 28
DURATION CODE? 3
FREQUENCY CODE? 21
DURATION CODE? 5
FREQUENCY CODE? 26
DURATION CODE? 3
FREQUENCY CODE? 28
DURATION CODE? 3
FREQUENCY CODE? 30
DURATION CODE? 3
FREQUENCY CODE? 26
DURATION CODE? 7
READY
```


21. MARK SIX

It will generate 6 random numbers with one extra special number.

```
10 REM MARK SIX
20 CLS
30 FOR I = 1 TO 7
40 A(I)=RND(36)
50 IF I = 1 THEN 90
60 FOR J = 1 TO I-1
70 IF A(I)=A(J) THEN 40
80 NEXT
90 NEXT
100 PRINT "THE NOS. ARE :"
110 FOR I = 1 TO 5
120 FOR J = 1 TO 5
130 IF A(J)<A(J+1) THEN 150
140 B=A(J) : A(J)=A(J+1) : A(J+1)=B
150 NEXT : NEXT
160 FOR J = 1 TO 6
170 PRINT A(J);
180 NEXT
190 PRINT
200 PRINT "SPECIAL NO. IS :"
210 PRINT A(7)
220 END

RUN

THE NOS. ARE :
 4  17  23  30  33  34
SPECIAL NO. IS :
 9
READY
```

This page is intended to leave bank

This page is intended to leave bank

DICK SMITH ELECTRONICS

VZ300

PERSONAL COLOUR COMPUTER

MADE IN HONG KONG

Cat No. X7300

81-2006-10